2019

Algorithms for synteny-based phylostratigraphy and gene origin classification

Zebulun Arendsee
Iowa State University

Algorithms for synteny-based phylostratigraphy and gene origin classification

by

Zebulun Arendsee

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Bioinformatics and Computational Biology

Program of Study Committee:
Eve Syrkin Wurtele, Major Professor
Karin Dorman
Matthew Hufford
Dennis Lavrov
Basil Nikolau

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

To Fawn, Rain and River

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This work would not have been written without the support of Iowa State University, the Bioinformatics and Computational Biology program, and the Genetics Development and Cell Biology department. I specifically thank my advisor Eve Wurtele, for her support as I meandered through topics and her patient guidance as I learned to write papers, present my work, and navigate peer review. I am grateful to Karin Dorman for her assistance as my co-major advisor, thoughtful instruction as my statistics teacher, meticulous input as a co-author, and direction as a committee member. I thank those who served on my committee — Matthew Hufford, Dennis Lavrov, Basil Nikolau, Guru Rao, Beate Schmittmann, and Steven Cannon — for their many good questions and thoughtful advice. I thank my fellow labmates Priyanka Bhandary, Jing Li, Urminder Singh and Arun Seetharam for feedback on my often buggy tools and comments on my papers. I am grateful to Andrew Wilkey for helping me build the first synder prototype; to my officemate Manhoi Hur for all his advice and support; and to Jennifer Chang for proofreading of my papers and listening to my rants. I thank Trish Stauble for all her help from my first day as a prospective student until her retirement. I also thank Fawn for staying with me through the seemingly endless years of poverty and stress.

ABSTRACT

With every newly sequenced species we discover hundreds of novel protein coding genes. Many of these "orphan" genes have been experimentally proven to have dramatic functions in development, sexual dimorphism, pathogen resistance, and social traits like symbiosis. Whereas in the past, researchers viewed genes as the product of continuous variation acting on ancient material, we now know that novel genes may arise *de novo* from non-genic sequence. Thus evolutionary experimentation is not limited to tweaking existing genes or their regulatory patterns. Any orphan genes that arose in the distant past, should appear today as lineage-specific genes (or gene families). The search for genes by their relative time of origin is called "phylostratigraphy". However, phylostratigraphy has proven to be a challenging task with different methodologies often yielding contradictory conclusions. Standard phylostratigraphy infers the age of a gene by finding the most distant species that has an inferred homolog. However, this approach is highly sensitive to annotation quality and cannot easily distinguish between rapidly evolving genes and genes of *de novo* origin.

This dissertation contributes a suite of tools for more accurately determining the phylostratigraphic age of genes and the level of support for the classification. First, we developed `phylostratr` to automate standard phylostratigraphy. Second, we developed a program, `synder`, to infer syntenic-homologs of query features using a synteny map. Third, we developed `fagin`, a package that builds on `synder` to search query genes against related species for traces of genic or non-genic orthology. The pipeline can distinguish orphans with high-confidence data support from orphans identified due to bad assembly or missing data. We traced many orphans to their non-genic cousins, identifying the

non-genic footprint from which they arose. We linked others to putative genes in related species from which they diverged beyond recognition. Knowing the approximate location of each gene across species and the amount of data support provides a launching point for future orphan studies.

CHAPTER 1. GENERAL INTRODUCTION

The central contribution of this thesis is a system of tools and methods for reproducible analysis of orphan genes and phylostratigraphy. The second chapter, apart from being a literature review, includes a phylostratigraphy study of *Arabidopsis thaliana*. The third chapter introduces `phylostratr`, an R package that automates phylostratigraphy and provides many diagnostics. The sixth chapter introduces `fagin`, which augments standard methods of phylostratigraphy with syntenic information and a deep analysis of possible sources of error. Chapters four and five introduce two tools that are used by `fagin` (see **Figure 1.1**): a synteny-based ortholog prediction program (`synder`) and a monadic pipeline program (`rmonad`). The following paragraphs further describe the progression of ideas across the chapters.

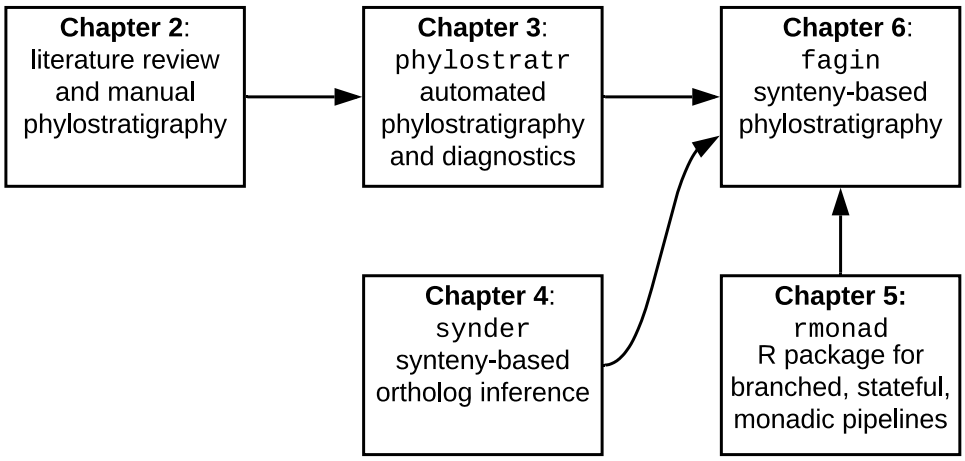


Figure 1.1 Organization of chapters. Chapter 6 follows logically from Chapter 3, but is dependent on the products of Chapters 4 and 5.

The **second chapter** is a literature review of orphan genes and phylostratigraphy with a focus on plants [1]. This chapter also contains a phylostratigraphic study of the protein-coding genes in *Arabidopsis thaliana* and study of their trends over time. Determining the phylostrata of a gene requires finding the most distant clade that contains a protein with statistically significant similarity. Since the phylostata inferences are dependent only on the single most distant similar gene, the method is highly sensitive to false positives. Thus, to achieve a balance between good representation of proteomes in each clade branching away from the focal species (*A. thaliana*) and reduce the risk of false positives, I hand-selected proteomes that were 1) of high-quality and 2) as widely separated on the clade tree as possible.

The **third chapter** introduces `phylostratr` [2] which is an R package we developed to serve as a general framework for conventional phylostratigraphy. The key contributions of this chapter are 1) an algorithm for automatically selecting diverse species from a tree while respecting a vector of quality scores give for each species, 2) full, reproducible automation of phylostratigraphy (species selection, proteome retrieval, sequence search, phylostratification), 3) a suite of diagnostics for identifying common problems with input data (such as missing organelle genomes) and for assessing the quality of genomes. This chapter further discusses the limitations of standard phylostratigraphy (including developments in the years since the publication of the second chapter) and introduces the need for augmenting phylostratigraphy with syntenic information.

The **fourth chapter** introduces `synder` [3], a tool for the high-performance tracing of features between genomes based on a synteny map. `synder` allows a feature (such as an orphan gene) to be traced to inferred orthologous genomic regions (the "seach intervals") in related species. This approach does not require any sequence similarity between the query and the target, thus is can be used to search for orthologs of features that are too small, poorly conserved, or numerous for purely similarity-based inference tools to be

effective. `synder` is the foundation of the synteny-based phylostratigraphy introduced in chapter 6.

The **fifth chapter** introduces `rmonad` [4], a monadic pipelines program for R (available on CRAN where it has been downloaded over 3500 times). Throughout my graduate program I have been searching for more elegant methods to handle the complexity of data analysis in bioinformatics (and beyond). I designed `rmonad` to act as a hybrid pipeline system, data management system, and documentation system. It supports large branching pipelines with thousands of nodes, captures all raised warnings/messages/errors, provides automatic benchmarking, stores documentation and data summarized data, and preserves the history of all past operations as a directed graph. Thus everything needed for report generation and debugging of a pipeline is wrapped into a single, self-describing object. `rmonad` is the pipeline manager for `fagin`, the topic of the sixth chapter.

The **sixth chapter** introduces `fagin`, a tool that performs synteny-based analysis of the origins of genes across a gene family. `fagin` builds on the synteny analyzing functions of `synder` and is implemented as an `rmonad` pipeline. `fagin` uses `synder` to define a narrow genomic search space for each query gene, then `fagin` mediates an in depth search of this region to decide whether 1) the query gene has a potentially coding ortholog, 2) the query gene has a DNA match to something that is not coding (consistent with the hypothesis that the query is a de novo orphan), or 3) that *no answer can be found*. By including the third, undecidable, category, `fagin` distinquishes between query genes that appear to be lineage-specific (e.g. orphans) because of 1) missing data (e.g. bad genome assemblies) or ambiguous biological outcomes (e.g. indels) or 2) because the ortholog of the query is non-genic in the target. The homolog classes can be integrated across the tree to determine a phylostratum for each gene. `fagin` makes decisions based on a decision tree that can be modified to add new types of data to the decisions. We anticipate `fagin` will serve in the future as a general framework for phylostratigraphy and orthology inference.

The appendices contain two relevant side-projects.

The first side-project is my investigation into an odd family of genes I named the ICEHGR family (after their conserved motif). This large family of genes is unique to just one species, and appears to have evolved through rapid duplication at both the gene level (generating all the members of the family) and at the sequence level (generating many tandem repeats of each ICEHGR motif).

The second side-project is the `metaoku` interactive visualization tool and the SAND data format. `metaoku` was designed to allow automatic visualization based on data type in up to three dimensions. SAND (Self-Annotating Nested Data) was a data format I was developing that was intended to formalize the norm of sharing data in nested folders of files. However, this project is now deprecated since the goal is mostly fulfilled by the Frictionless Data group and DataHub (a resource I use in both the `fagin` and `synder` case studies to distribute data).

There are many additional projects I have developed that where used in the dissertation research but not included in this dissertation, I will mention four here. First `smof` [5]: an elegant UNIX-style tool for exploring and analyzing FASTA sequence. I used this tool extensively throughout all of my bioinformatics research. Second `rhmmer` [6]: An R package for parsing HMMER output, used within `phylostratr`, that is available on CRAN (with nearly 2000 downloads). Third `onekp` [7]: An R package available through rOpenSci that allows remote access and search of the data available from the 1000 Plant Trascriptomes (1KP) project. This package was used in the `phylostratr` case study. Fourth `taxizedb` [8]: I am one of the coauthors of this rOpenSci package for accessing and exploring taxonomies (including the NCBI common tree). This package is foundational to the `phylstratr` program.

Overall, the contribution of this thesis is the development of a powerful, flexible, reproducible and extendable framework for orphan analysis, phylostratigraphy and more general orthology studies.

Bibliography

[1] Arendsee, Z.W. *et al.* (2014) Coming of age: orphan genes in plants. *Trends in plant science* 19, 698–708

[2] Arendsee, Z. *et al.* (2019) phylostratr: A framework for phylostratigraphy. *Bioinformatics*

[3] Arendsee, Z. *et al.* (2019) synder: inferring genomic orthologs from synteny maps. *bioRxiv*

[4] Arendsee, Z. and Chang, J. (2018) . arendsee/rmonad: rmonad v0.6.1

[5] Arendsee, Z. *et al.* (2018) . incertae-sedis/smof: smof v2.13.1

[6] Arendsee, Z. (2018) . arendsee/rhmmer: rhmmer v0.1.0

[7] Arendsee, Z. (2018) *onekp: remote access to the 1KP dataset*

[8] Chamberlain, S. *et al.* (2018) . ropensci/taxizedb: taxizedb v0.1.6

CHAPTER 2. COMING OF AGE: ORPHAN GENES IN PLANTS

Zebulun Arendsee, Ling Li, Eve Syrkin Wurtele

Modified from a paper published in *Trends in Plant Science*

Abstract

Sizable minorities of protein-coding genes from every sequenced eukaryotic and prokaryotic genome are unique to the species. These so-called 'orphan genes' may evolve de novo from non-coding sequence or be derived from older coding material. They are often associated with environmental stress responses and species-specific traits or regulatory patterns. However, difficulties in studying genes where comparative analysis is impossible, and a bias towards broadly conserved genes, have resulted in under-appreciation of their importance. We review here the identification, possible origins, evolutionary trends, and functions of orphans with an emphasis on their role in plant biology. We exemplify several evolutionary trends with an analysis of *Arabidopsis thaliana*.

2.1 All species have a cadre of unique genes

Until the past few years the consensus was that new genes arise via combinations of processes such as duplication, fusion, fission, and transposition of existing protein-coding genes. Fischer and Eisenberg noticed that all sequenced bacteria contained genes without detectable homologs in any sequenced relative [1]. They postulated this uniqueness was a real phenomenon, rather than an artifact of poor annotation or sparse sequencing among nearby species, as some claimed [2]. Since the advent of next-generation sequencing, the

analysis of a multitude of genomes has shown that such orphan genes are widespread across all domains of life [3, 4, 5] and viruses [6].

Continual genesis of novel genes in an organism where protein count is fairly constant implies equilibrium between gene origin and extinction. A reasonable hypothesis is that most of the turnover occurs in the youngest genes [7]. This has been demonstrated in Drosophila [8] and is reflected in the degree of conservation of existing genes in each genome. Under this model there is a vast, dynamic reservoir of novel genes. We will discuss: (i) the origin of orphans and their regulatory elements, (ii) their maturation into established genes, and (iii) the functions into which they are recruited.

2.2 Orphans originate in diverse ways

Orphans may be defined as genes with coding sequences utterly unique to the species; in other words, genes that produce previously non-existing (novel) proteins. They are a subset of taxonomically restricted (also called lineage-specific) genes that are specific to a particular taxon (e.g., malvid-specific or Brassicaceae-specific genes). Genes are generally classified as being orphans if they lack coding sequence similarity outside their species (usually quantified by BLAST). This classification method accepts as orphans, genes that are newly born from non-genic sequence, as well as descendants of ancient genes whose coding sequences have changed beyond recognition; it rejects horizontally transferred genes and duplicated genes that may have assumed a new function but whose proteins are still recognizable (i.e., "new" genes that are not orphans). Analysis of the genomic contexts and sequences of orphan genes can often reveal their origins, as reviewed in [7]. Some can be traced to highly divergent products of gene duplications, overlapping or anti-sense reading frames (overprinting), domesticated transposons, resurrected pseudogenes, or early frameshift mutations [8, 9, 10, 11, 12]. Others may arise de novo from non-coding sequence. Early doubts that protein-coding genes could spontaneously arise [13] have been put to rest by a flood of papers tracing orphans to their non-genic roots

[14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. A recent study suggests a continuum between very weakly transcribed and translated open reading frames (ORFs) and highly functional, mature genes [24]. Table 2 from [10] shows a cross-species, quantitative comparison of the origins of orphan genes. In *A. thaliana*, over half of the orphans appear to have arisen de novo, based on similarity to non-genic regions of *Arabidopsis lyrata* [9]. Estimates of the percentage of genes that are orphans in various species ranges wildly from $< 1 - 71\%$ [5, 9, 10, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33], with 5-15% being fairly typical [10, 31, 33]. A portion of this disparity is attributable to the varying evolutionary distance between each focal species and its nearest sequenced relatives [10]. Other sources of variation are the quality of the genome datasets and the methods used in orphan identification (e.g., three independent studies of *A. thaliana* report 958, 1324, and 1430 orphans, respectively [9, 34, 35]). However, much of the variation reported is likely due to real differences in evolutionary pressures and molecular genetic phenomena that are as yet unknown.

2.3 Phylostratigraphy classifies genes by age

Genes can be stratified by age via a technique known as phylostratigraphy that traces modern genes back to their orphan founders [36]. **Figure 2.1** shows a phylostratigraph of the protein-coding genes of *A. thaliana*. The general approach is to select hierarchical taxonomic groups ascending from the focal species, and for each gene find the oldest taxon in which it has a homolog [36]. By describing the characteristics of increasingly ancient phylostrata, the path from genomic noise to mature protein is revealed. Conventional phylostratigraphic analyses make two major assumptions. The first is that simple search algorithms (such as protein-BLAST) are adequate for the identification of distant homologs. This assumption is supported by evolutionary simulations [37]. However, a recent study of viral orphan genes that used more sensitive algorithms (PSI-BLAST, HHBlits, and HHPred) predicted homologs for about a quarter of genes that had been identified as genus-specific by protein-BLAST [38]. A second assumption is that the old-

est components of genes are no older than the gene founders. This assumption is violated if an old domain or exon is incorporated into a young protein. This issue has been noted previously, but was considered not to be a serious impediment, at least not in metazoans [7]. A recent review acknowledges the successes of phylostratigraphy [36, 39, 40] but argues that phylogenetic reconciliation methods offer a more nuanced understanding of the events underlying gene histories [41].

What are the prospects of a young orphan gene? Phylostratigraphic analyses indicate that some orphans survive to fixation. These manifest as gene families that are taxonomically restricted to the clade descending from the species in which they arose. Genes from older phylostrata tend towards greater length, complexity, and connectivity. **Figure 2.2** includes an overview of several cross-phylostrata traits in *A. thaliana* genes, and compares them to non-genic ORFs.

2.4 Orphans mature with time

A steady, several-fold increase in protein length from species-specific genes to universally conserved genes has been noted in several metazoans [11, 42], yeast [24], and *A. thaliana* [35] (**Figure 2.2**A). This is largely due to an increase in the number of exons because the average exon length is somewhat constant, as seen in metazoa [11] and in the older *A. thaliana* phylostrata (**Figure 2.2**B). Although in some species (such as rice, zebrafish, and humans) genes from recent phylostrata have particularly long exons [11, 43], in *A. thaliana* there is a significant increase in exon size (about twofold) across the first several phylostrata.

By several criteria, younger genes are more random and specialize over time. For example, amino acid composition bias increases with age in yeast [24] and many bacteria [44]. Similarly, codon bias, which is often used as a proxy for translational optimization [45], increases with gene age in primates [42], yeast [24], and Drosophila [8]. Percent GC content also increases gradually across the phylostrata for a number of species [9, 10],

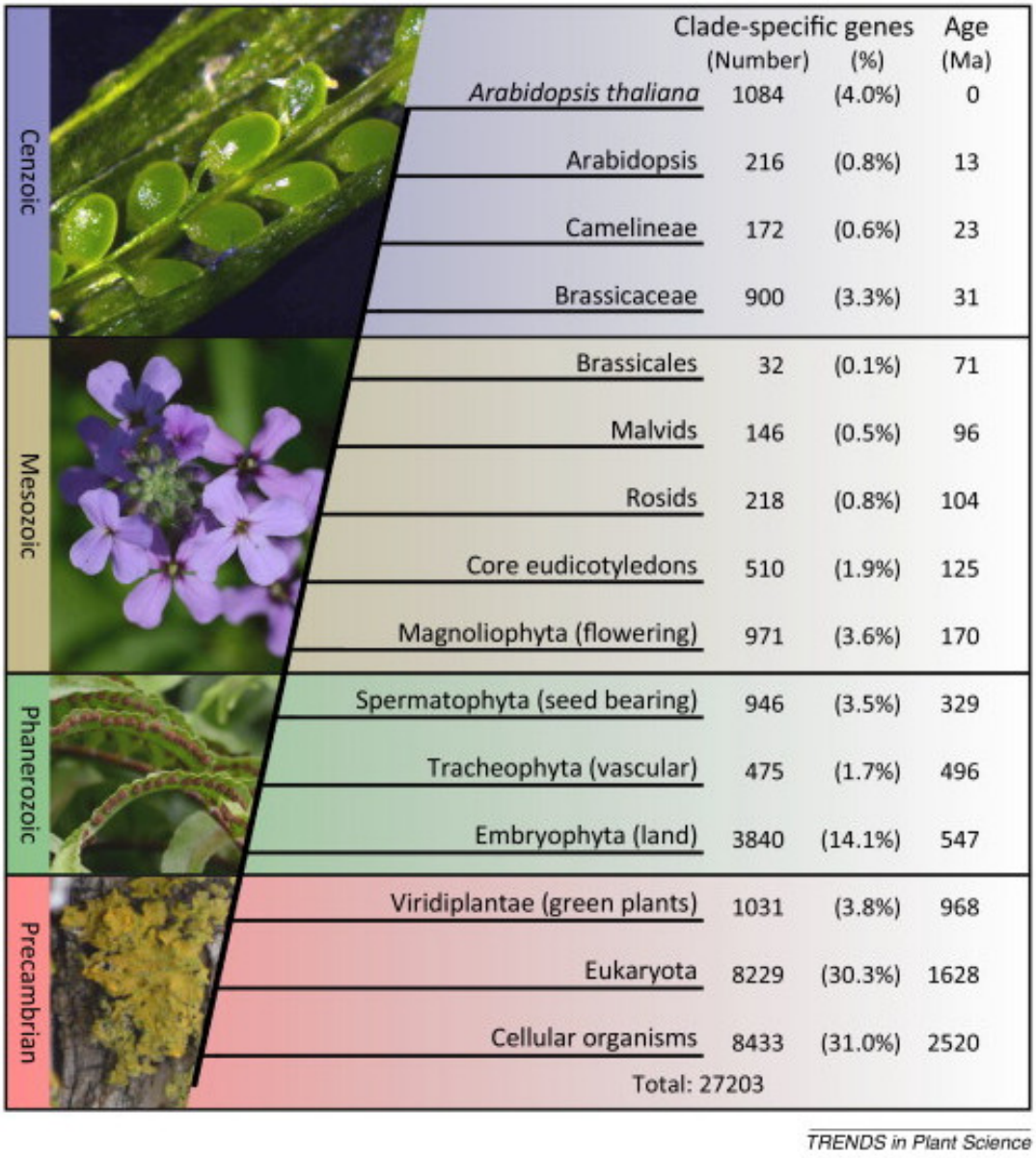


Figure 2.1 Age stratification of all genes in *Arabidopsis thaliana*. Each gene is assigned to the oldest clade (or "phylostratum") that contains a homolog, as inferred by a protein-BLAST of each *A. thaliana* gene against a selected set of genomes (Table S1 in the supplementary material online) with a threshold e value of 10^{-5}. All *A. thaliana* genes are from the TAIR10 release (in cases where one locus has multiple gene models, we used the first). Organelle genomes were not included in the analysis. Age is in Ma (millions of years ago) and refers to the estimated time since the diversification of the clade from its most recent common ancestor. For references for the age assignments and a list of genomes searched in each phylostratum, see **Table 2.1**. For complete searchable phylostratum assignments, see AtGeneSearch (http://www.metnetdb.org/MetNet_atGeneSearch.htm).

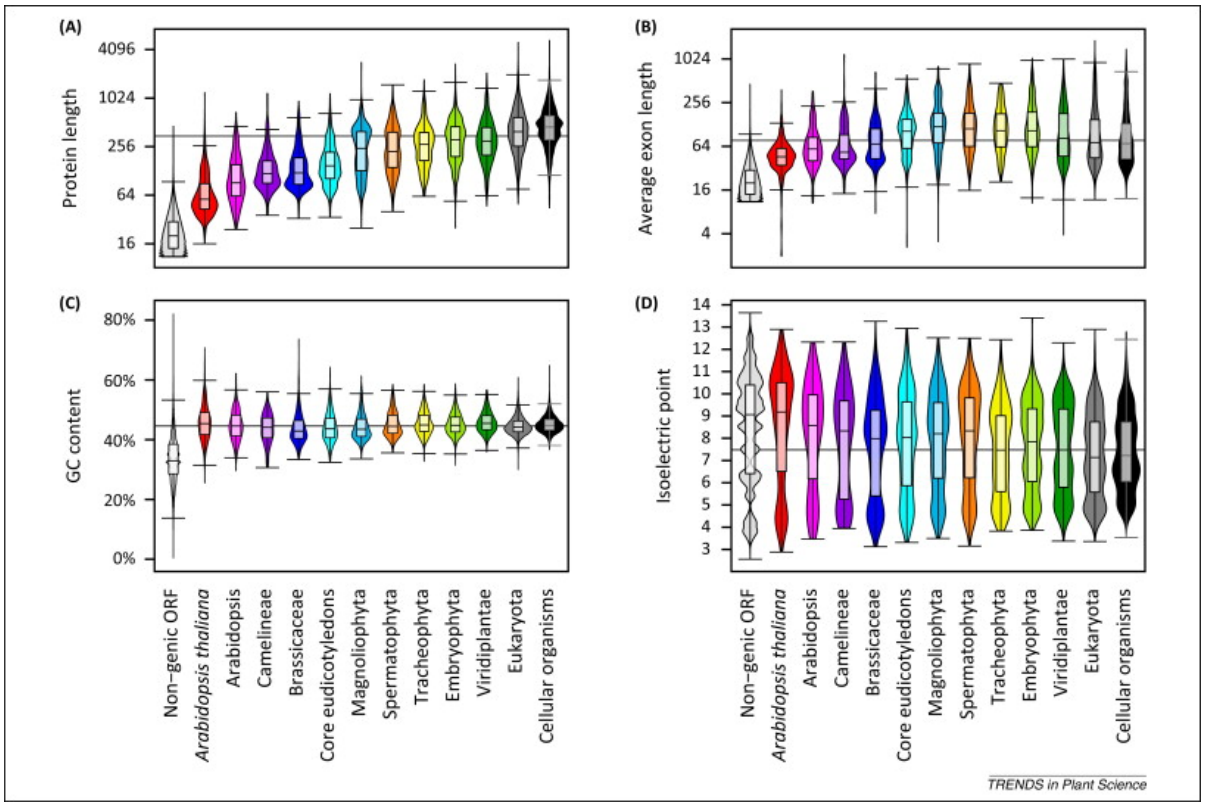


Figure 2.2 Trends across *A. thaliana* phylostrata and non-genic open reading frames (ngORFs). All ORFs in *A. thaliana* were predicted with the getorf program from the EMBOSS toolkit (version 6.4.0.0). All ORFs that overlapped (on either strand) a protein-coding sequence, transposable element, or pseudogene (all as annotated in the TAIR10 genome release) were then removed, to create the ngORF set. Each "violin" plot represents the range of values (vertically, using a box plot) and density of genes at that value (horizontally, thickness being proportional to the percent of genes at each value). Gray horizontal lines mark median values across all genes (excluding ngORFs). Strata outgroups are detailed in **Table 2.1**. (A) Average protein length (amino acids). (B) Average exon length (codons) (all ngORFs have one exon); the fairly constant exon length reported for metazoa [11] contrasts with the upward trend in the youngest few phylostrata of *A. thaliana* (twofold increase to eudicotyledons stratum, p-value $< 10^{-15}$). (C) Percent GC content; the content of the ngORFs contrast starkly with that of all the genes, which increases slightly, but significantly (p-value $< 10^{-15}$), from 43.3% in orphans to 44.8% in cellular organisms. (D) Predicted isoelectric point (pI); the difference in mean pI between orphans and cellular organisms is highly significant (p-value $< 10^{-16}$). For searchable classification of individual genes by phylostratum, see AtGeneSearch (http://www.metnetdb.org/MetNet_atGeneSearch.htm).

including *A. thaliana* [34]. Even the youngest genes in *A. thaliana*, however, are sharply separated in GC content from the pool of non-genic ORFs (43% median GC content for orphans versus 32% for non-genic ORFs) (**Figure 2.2**C).

Protein isoelectric points across *A. thaliana* phylostrata tend to decrease from a median of around pI 9.1 for nongenic ORFs and orphans to pI 7.2 for genes in the oldest stratum (**Figure 2.2**D). In yeast, protein aggregation propensity decreases with age [46] (protein aggregates are often toxic, as exemplified by their role in Alzheimer's disease [47]).

Younger genes also appear to be more evolutionarily radical. Microsatellites and low-complexity regions are more common in younger genes of *Drosophila melanogaster* [8], rice (*Oriza sativa*) [27], and mammals [48]. These regions can be powerful drivers of evolution [49]. For example, expansion of a dinucleotide repeat in an Antarctic fish provided the material for the evolution of a novel 700 antifreeze protein [50]. The robustness of protein secondary structure to mutation also increases with age in yeast [46]. High robustness decreases the likelihood of radical structural change, but enhances the subtle evolution of mature proteins by preserving their primary function while they safely explore novel ones [51]. Conversely, low robustness favors more radical changes but heightens the risk of evolutionary failure.

The prospects of a young gene are dependent on how securely it can integrate itself into vital processes or networks. Studies in yeast suggest transcriptional regulation appears very quickly, but protein-protein and genetic (or epistatic) interactions develop more gradually [46]. Orphan genes tend to be less expressed [8, 24], and in a narrower range of tissues [9, 42]. A comparison of the protein-protein interactions of de novo orphan genes to those of recently duplicated genes found that young de novo genes were very poorly connected relative to young duplicated genes [52]. However, genes that are predicted to have originated de novo more than 100 million years ago are as well connected as their duplicated peers [52].

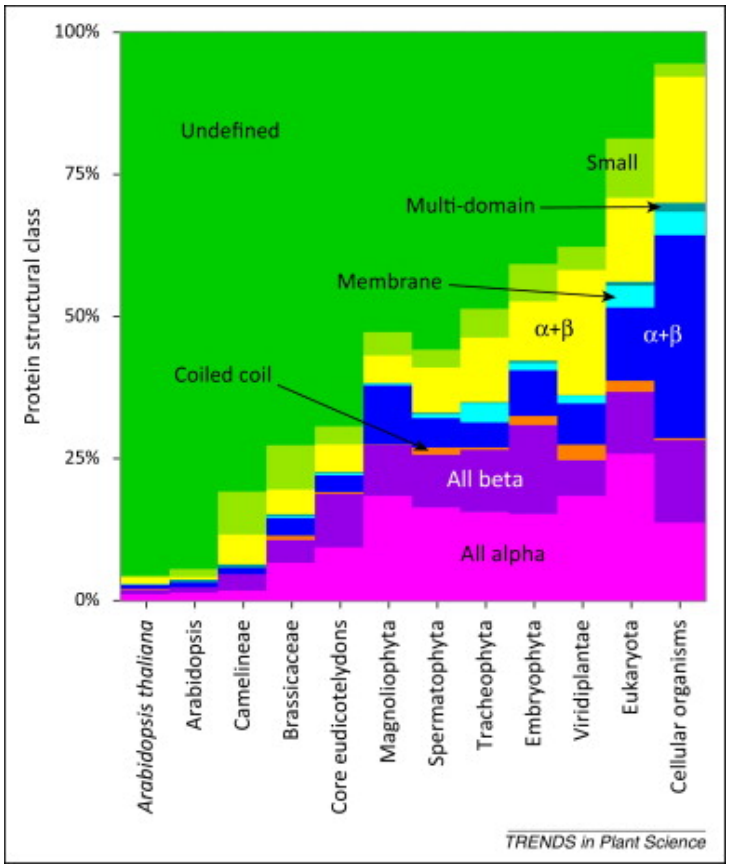


Figure 2.3 Predicted protein structural classes for *Arabidopsis thaliana* proteins are assigned to each phylostratum. Predictions are based on a search against the SCOP (Structural Classification of Proteins) database, which we retrieved from the TAIR website (http://www.arabidopsis.org/tools/bulk/protein/index.jsp).
`All beta` and `All alpha` refer to proteins composed mostly of α-helices or of β-sheets, respectively. The "α/β" and "$\alpha+\beta$" classes consist of mixtures of α-helix and β-sheet domains. The `small` class contains structures centered around a metal ligand, heme, or a disulfide bridge. `Multi-domain` proteins have domains from more than the one structural class. `Membrane` indicates a predominance of membrane-spanning domains. Proteins classified as `undefined` have no detectable similarity to any structure in SCOP (this does not necessarily imply they are unstructured or disordered). See supplementary data online. For individual protein assignments by phylostratum, see AtGeneSearch (http://www.metnetdb.org/MetNet_atGeneSearch.htm)

There are several non-exclusive mechanisms by which orphans may gain regulatory elements. Transposon insertion upstream of the transcription start-site can both increase expression overall and couple expression to specific stresses [53] or tissues (reviewed in [54]); this mechanism could create a particularly dramatic effect on expression of de novo orphans, which are usually less regulated to begin with. Some orphans may share regulatory elements with older genes via gene overlap [21], association with a bidirectional promoter [11], or by being located within an intron [21]. Orphans that are derived from coding gene duplication followed by rapid mutation [55], or from a pseudogene (e.g., by an early frameshift mutation), may inherit some regulatory elements from their prior context. Alternatively, cryptic regulatory sites may predate the origin of the gene or arise de novo later [56]. Finally, regulation may arise via epigenetics, as is illustrated for the QQS (QUA-QUINE STARCH, AT3G30720) orphan of *A. thaliana* [57].

2.5 Orphans are often functional and sometimes essential

It might be expected that recently evolved genes would not be crucial for survival; after all, the organism seemed to do quite well without them. However, although the function of the vast majority of individual orphans is unknown, and while they generally lack identifiable folds (**Figure 2.3**), functional motifs [27] and recognizable domains [11], there is ample evidence of widespread functionality. A study of six de novo genes in Drosophila found four to be essential (i.e., homozygote nulls are embryo-lethal) [58]. Another Drosophila study that included 16 randomly selected de novo orphans found that three of these orphans were essential [59]. However, in mice and yeast genes from older phylostrata are vastly more likely to be essential then their younger counterparts [60]. Although particular plant orphans have been shown to be important for survival under specific conditions [61], none have yet been reported to be embryo-lethal if eliminated. Further evidence for orphan functionality can be gleaned from the widespread reports that purifying selection is high in old genes and positive selection is high in younger ones

[10, 24, 35, 62]. Drosophila orphans were found to be under purifying selection (though much weaker than in older genes) [8]. Both positive and purifying selection generally implies functionality (as opposed to neutral drift).

2.6 Orphans are important for responses to environmental stresses

There is evidence that orphans are recruited into roles regulating responses to the ever varying environment [4]. Orphans are preferentially expressed in reaction to abiotic stresses in yeast [24], the water flea [30], *A. thaliana* [9, 63], and rice [27]. Several orphans appear to play a role in plant responses to the environment [4, 60, 63, 64, 65]. A recent study revealed that more than 80% of knockout mutants of genes of unknown function in *A. thaliana* had an altered phenotype when subjected to stress, conferring either protection against, or acting as suppressors of, various abiotic stresses, especially oxidative and osmotic stresses [61]. Of the 91 orphan genes included in [61] (as identified by using the phylostrata assignments of **Figure 2.1**), 76 exhibit at least one stress-related phenotype.

Orphans and taxonomically restricted genes from a variety of plants and invertebrates play a role in biotic interactions such as in inter-organism defense, attack, and signaling. Many of these lack trans-species sequence similarity, but are not of recent de novo origin. These "ageless" orphans tend to duplicate, rearrange, and mutate very quickly, but their general function and secondary structure is conserved. The most common class of these is a subset of the small, cysteine-rich, secreted proteins known as CRISPs (or CRPs) [66]. The distinguishing characteristic of CRISPs is a set of conserved cysteines that forms the backbone of a hydrophobic core, which protects them from proteases and increases their functional lifespan [67]. The remainder of the sequence is often free to diverge. This, coupled with their short length, could allow them to quickly mutate beyond recognition. Many other CRISPs are members of large, recognizable families [66, 67, 68], and thus

are not orphans, demonstrating that small size alone does not explain the many CRISPs that are predicted to be orphans. The origin of CRISPs has not been comprehensively investigated.

Plants and their pathogens and pests are locked in a chemical arms race. Fungal pathogens rely on secreted effector molecules tailored to a particular plant species or even to a specific tissue (reviewed in [69]). CRISPs are found in almost all pathogenic fungi and very often have no identifiable homologs outside their species [70, 71]. Indeed, fungal effectors are so prone to being species-specific that those with identifiable homologs are noteworthy [72, 73]. Invertebrate pests also rely on lineage-specific protein effectors. Aphids secrete a soup of proteins into their hosts as they feed, about half of which are aphid-specific [74]. A genus-specific pathogenesis effector, MAP-1 (*Meloidogyne avirulence* protein-1), has been reported in the nematode, Meloidogyne [75].

Far from being defenseless, plants also recruit orphans into this war. A 38 amino acid antifungal CRISP (Ps-AFP1) with a unique fold and no extra-species similarity was recently discovered in the garden pea *Pisum sativum* [76]. The rice-specific, de novo orphan OsDR10 is a suppressor of the pathogen-induced defense response [77]. The 49 amino acid Brassicaceae-specific antifungal peptide EWR1 (ENHANCER OF VASCULAR WILT RESISTANCE 1, AT3G13437) confers fungal resistance when transferred to *Nicotiana benthamiana*, a close relative of tobacco [78]. The full extent of the orphan contribution to plant defense is unknown.

Orphans are also common in the toxic peptide armsraces of cnidarians (jellyfish and sea anemones). Of the 20 toxins produced by the sea anemone *Nematostella vectensis*, six are species-specific and eight are cnidarian-specific [79]. The 40 amino acid, antimicrobial CRISP aurelin has no apparent homologs outside the jellyfish *Aurelia aurita* [80]. It is structurally similar to many animal toxins, but whether these are true homologs or the result of convergent evolution is uncertain [81]. Aurelin is only one example from a large

group of antimicrobial CRISPs that are often restricted to a single species or narrow clade [82].

2.7 An association of orphans with reproduction

Some orphans are highly expressed in structures associated with reproduction and early development. In Drosophila (but not human [43]), orphans are highly over-represented in the testes [14, 58, 83]. QQS is one example of an orphan that is differentially expressed in the pollen [84, 85], but whether orphans in plants generally follow the Drosophila trend is unknown.

Young genes are highly expressed in early and late embryological development whereas ancient genes dominate mid-embryogenesis. This embryonic "hourglass" phenomenon [86] is manifest in both metazoans [87] and plants [88], despite their independent rise to multicellularity [89], suggesting a fundamental significance.

2.8 Orphans are often recruited into lineage-specific structures

A unique structure of the cnidarian phylum is the nematocyte. When stimulated, nematocytes can hurl threads that can act as potentially poisonous harpoons, grapples to draw in prey, or tethers to attach to surfaces [90]. In *Hydra magnipapillata* (the model medusozoan), 41 of 50 nematocyte-specific genes [91] are cnidarian-specific. This is consistent with the hypothesis that orphans were heavily recruited in the origin of this novel cell type [3].

An aphid-specific cell type that nurtures nitrogen-fixing bacteria is heavily enriched in secreted orphan genes [92]. Many secreted, proline-rich tandem repeat proteins (TRPs), which are important in cell wall function in plants, are family-specific and may be involved in novel adaptations of specialized structures such as those needed for bacterial symbiosis in legumes and seed storage vacuoles in grasses [93].

Table 2.1 Phylostratum age estimates and representatives

Phylostratum	**Time (mya)**	**Outgroup representatives**
Arabidopsis	13 ± 5 [94]	*Arabidopsis lyrata*
Camelineae	23 ± 5 [94]	*Capsella rubella*
Brassicaceae	31 ± 5 [94]	*Brassica rapa*, *Thellungiella halophila*
Brassicalesa	71 ± 11 [94]	*Carica papaya*
Malvids	96 ± 13 [95]	*Citrus clementine*, *Citrus sinensis*, *Eucalyptus grandis*, *Gossypium raimondii*, *Theobroma cacao*
Rosids	104 ± 3 [95]	*Cannabis sativab*, *Populus trichocarpa*, *Vitis vinifera*
Core eudicotyledons	125 ± 3 [96]	*Camptotheca acuminatab*, *Echinacea purpureab*, *Solanum lycopersicum*
Magnoliophyta	170 ± 3 [96]	*Brachypodium distachyon*, *Dioscorea villosab*, *Musa acuminate*, *Oryza sativa*, *Zea mays*
Spermatophyta	329 ± 3 [97]	*Ginkgo bilobab*
Tracheophyta	496 ± 39 [98]	*Selaginella moellendorffii*
Embryophyta	547 ± 44 [98]	*Physcomitrella patens*
Viridiplantae	968 ± 93 [99]	*Chlamydomonas reinhardtii*, *Micromonas pusilla*, *Ostreococcus lucimarinus*
Eukaryota	1609 ± 60 [99]	*Bigelowiella natans*, *Cyanidioschyzon merolae*, *Dictyostelium discoideum*, *Giardia intestinalis*, *Mus musculus*, *Phaeodactylum tricornutum*, *Plasmodium falciparum*, *Saccharomyces cerevisiae*
Cellular organisms	2460 ± 140 [100]	*Acidilobus saccharovorans*, *Arthrospira platensis*, *Bacillus subtilis*, *Ferroplasma acidarmanus*, *Gloeobacter violaceus*, *Haloquadratum walsbyi*, *Methanopyrus kandleri*, *Nostoc punctiforme*, *Prochlorococcus marinus*, *Pseudomonas aeruginosa*, *Pyrobaculum islandicum*, *Salinispora arenicola*, *Sulfolobus islandicus*
Root	3480 [101]	None

2.9 Orphans are seldom (if ever) recruited to catalytic roles

Perhaps the most sophisticated of protein functions is catalysis, requiring highly specialized structures to interact specifically with substrates. If enzymatic capacity were easily evolved, one might postulate that the most natural place to find young genes with catalytic competence would be in specialized metabolic pathways. Plants augment their core metabolism with a dazzling array of clade-specific specialized compounds. The enzymes catalyzing these unique pathways tend to have lower substrate specificity and efficiency than the ancient core enzymes [102], which might be interpreted as having a less sophisticated structure. However, the enzymes of specialized metabolism that have been studied can be traced to preexisting proteins that are relaxed duplicates of enzymes of core metabolism [102]. The single known exception is chalcone isomerase, which can be traced back to an ancient metabolite binding protein of apparently prokaryotic origin [103].

To evaluate the origin of enzymes, we searched the *A. thaliana* database AraCyc (Release 11.5: https://www.arabidopsis.org/biocyc/), one of the most complete sets of metabolic pathways for any organism. AraCyc includes genes encoding catalytic and non-catalytic subunits of enzymes of both core and specialized metabolism — and we supplemented this search with human curation to minimize mis-annotation. The youngest gene encoding any catalytic subunit appears in the ancient Embryophyta phylostratum, and the vast majority can be traced back to a common origin in cellular organisms. Indeed, only a handful of genes postulated as encoding non-catalytic subunits appear to have arisen prior to Embryophyta. Because of the potential bias towards underestimating the age of genes in the deeper phylostratum using current methodologies, more detailed analysis may reveal that the origins of catalytic functions are even more ancient. The highly conserved core catalytic domains of enzymes, optimized by a billion years of evolution, can be thought of as the molecular equivalents of clockwork. Orphans,

by contrast, are more like bricks; they may have essential functions, but they are not sophisticated machines.

The frequent emergence in evolution of novel specialized biochemical pathways [102, 103, 104, 105, 106, 107, 108, 109] would require concomitant emergence of new regulatory elements; we postulate that it is here, rather than enzymes per se, that orphan genes may play an as yet undiscovered role in metabolism.

2.10 Concluding remarks and future directions

Orphans are evolutionary pioneers, freely exploring new protein space with little constraint or bias inherited from parents. They may be too young to compete in the catalytic arena, but in niches where the rules are in flux, such as interactions with other organisms and the varying environment, their youth may be a boon. Although the mystery of their origins is gradually being unraveled, the breadth and especially the mechanisms of their functions are poorly understood (Box 1). Nevertheless, one thing is certain; they are not mere curiosities of importance limited to rare cases in narrow clades. There are ample reasons to invest in our understanding of orphans.

First, orphans can be considered as the fodder for new proteins. As such, orphans and subsequent early phylostrata are ripe for study by structural biologists because they hold potential to reveal mechanisms of the origin of protein structural domains.

Second, the general principles governing orphan genesis, maturation, and functional recruitment gleaned from a model species, such as *A. thaliana*, will illuminate studies involving orphans in other species. An orphan aware experimentalist would know to expect orphans (and carefully consider any genome annotation she is relying on) when analyzing data relevant to stress or species-specific adaptations. She would also know to be skeptical, but very excited, if she found a catalytically active orphan.

Third, functionality of an orphan may be transferable to ectopic species, in other words species in which that particular orphan gene is not present. This has been demon-

strated for QQS in a metabolic context [84] and EWR1 in a disease resistance context [78]. If, as postulated in [84], many orphans rely on interactions with more conserved cellular components, then it is likely that they can function cross-species. Therefore, orphans may comprise a vast reservoir of functional proteins that can be tapped in the engineering of valuable cultivars and the discovery of novel peptide drugs and pesticides.

Fourth, understanding the basis for de novo gene evolution has major implications for synthetic biology. It will inspire novel approaches such as evolving genes with new function by random insertion of artificial orphan like sequences followed by selection for desired traits.

The cornucopia of ecotype-level genome and transcriptome data available for *A. thaliana* [110, 111, 112], and that gathered for other species, can be leveraged to map precisely the origins of orphans, the development of their regulatory elements, and all the diversity in orphan populations missed in a single reference genome. These data will allow the evolutionary pressure acting on species-specific genes to be quantified, differentiating between fast track, potentially functional orphans and their drifting counterparts. This will permit a deeper understanding of the factors that determine orphan birth, death, or integration into the panoply of conserved genes.

Bibliography

[1] Fischer, D. and Eisenberg, D. (1999) Finding families for genomic ORFans. *Bioinformatics (Oxford, England)* 15, 759–762

[2] Casari, G. *et al.* (1996) Bioinformatics and the discovery of gene function. *Trends in Genetics* 12, 244–245

[3] Khalturin, K. *et al.* (2009) More than just orphans: are taxonomically-restricted genes important in evolution? *Trends in Genetics* 25, 404–413

[4] Gollery, M. *et al.* (2006) What makes species unique? The contribution of proteins with obscure features. *Genome biology* 7, R57

[5] Wilson, G.A. *et al.* (2005) Orphans as taxonomically restricted and ecologically important genes. *Microbiology* 151, 2499–2501

[6] Yin, Y. and Fischer, D. (2008) Identification and investigation of ORFans in the viral world. *BMC Genomics* 9, 24

[7] Tautz, D. and Domazet-Lošo, T. (2011) The evolutionary origin of orphan genes. *Nature Reviews Genetics* 12, 692–702

[8] Palmieri, N. *et al.* (2014) The life cycle of Drosophila orphan genes. *Elife* 3

[9] Donoghue, M.T. *et al.* (2011) Evolutionary origins of brassicaceae specific genes in *Arabidopsis thaliana*. *BMC evolutionary biology* 11, 47

[10] Wissler, L. *et al.* (2013) Mechanisms and dynamics of orphan gene emergence in insect genomes. *Genome Biology and Evolution* 5, 439–455

[11] Neme, R. and Tautz, D. (2013) Phylogenetic patterns of emergence of new genes support a model of frequent de novo evolution. *BMC genomics* 14, 117

[12] Brosch, M. *et al.* (2011) Shotgun proteomics aids discovery of novel protein-coding genes, alternative splicing, and "resurrected" pseudogenes in the mouse genome. *Genome Research* 21, 756–767

[13] Jacob, F. (1977) Evolution and tinkering. *Science* 196, 1161–1166

[14] Levine, M.T. *et al.* (2006) Novel genes derived from noncoding DNA in Drosophila melanogaster are frequently X-linked and exhibit testis-biased expression. *Proceedings of the National Academy of Sciences* 103, 9935–9939

[15] Begun, D.J. *et al.* (2006) Evidence for de novo evolution of testis-expressed genes in the *Drosophila yakuba/Drosophila erecta* clade. *Genetics* 176, 1131–1137

[16] Cai, J. *et al.* (2008) De novo origination of a new protein-coding gene in *Saccharomyces* cerevisiae. *Genetics* 179, 487–496

[17] Li, D. *et al.* (2010) A de novo originated gene depresses budding yeast mating pathway and is repressed by the protein encoded by its antisense strand. *Cell research* 20, 408–420

[18] Li, C.Y. *et al.* (2010) A human-specific de novo protein-coding gene associated with human brain functions. *PLoS Computational Biology* 6, e1000734

[19] Knowles, D.G. and McLysaght, A. (2009) Recent de novo origin of human protein-coding genes. *Genome Research* 19, 1752–1759

[20] Heinen, T.J. *et al.* (2009) Emergence of a new gene from an intergenic region. *Current Biology* 19, 1527–1531

[21] Murphy, D.N. and McLysaght, A. (2012) De novo origin of protein-coding genes in murine rodents. *PloS one* 7, e48650

[22] Yang, Z. and Huang, J. (2011) De novo origin of new genes with introns in *Plasmodium vivax*. *FEBS Letters* 585, 641–644

[23] Zhao, L. *et al.* (2014) Origin and spread of de novo genes in *Drosophila melanogaster* populations. *Science* 343, 769–772

[24] Carvunis, A.R. *et al.* (2012) Proto-genes and de novo gene birth. *Nature* 487, 370–374

[25] Ekman, D. and Elofsson, A. (2010) Identifying and quantifying orphan protein sequences in fungi. *Journal of Molecular Biology* 396, 396–405

[26] Ye, C.Y. *et al.* (2013) Evolutionary analyses of non-family genes in plants. *The Plant Journal* 73, 788–797

[27] Guo, W.J. *et al.* (2007) Significant comparative characteristics between orphan and nonorphan genes in the rice (*Oryza sativa L.*) genome. *Comparative and Functional Genomics* 2007, 1–7

[28] Yang, L. *et al.* (2013) Genome-wide identification, characterization, and expression analysis of lineage-specific genes within zebrafish. *BMC genomics* 14, 65

[29] Hahn, M.W. *et al.* (2007) Gene family evolution across 12 Drosophila genomes. *PLoS Genetics* 3, e197

[30] Colbourne, J.K. *et al.* (2011) The ecoresponsive genome of *Daphnia* pulex. *Science* 331, 555–561

[31] Ohm, R.A. *et al.* (2012) Diverse lifestyles and strategies of plant pathogenesis encoded in the genomes of eighteen Dothideomycetes fungi. *PLoS Pathogens* 8, e1003037

[32] Gibson, A.K. *et al.* (2013) Why so many unknown genes? Partitioning orphans from a representative transcriptome of the lone star tick *Amblyomma americanum*. *BMC genomics* 14, 135

[33] Kuo, C.H. and Kissinger, J.C. (2008) Consistent and contrasting properties of lineage-specific genes in the apicomplexan parasites Plasmodium and Theileria. *BMC evolutionary biology* 8, 108

[34] Lin, H. *et al.* (2010) Comparative analyses reveal distinct sets of lineage-specific genes within *Arabidopsis thaliana*. *BMC Evolutionary Biology* 10, 41

[35] Guo, Y.L. (2013) Gene family evolution in green plants with emphasis on the origination and evolution of *Arabidopsis thaliana* genes. *The Plant Journal* 73, 941–951

[36] Domazet-Lošo, T. *et al.* (2007) A phylostratigraphy approach to uncover the genomic history of major adaptations in metazoan lineages. *Trends in Genetics* 23, 533–539

[37] Albà, M.M. and Castresana, J. (2007) On homology searches by protein blast and the characterization of the age of genes. *BMC evolutionary biology* 7, 53

[38] Kuchibhatla, D.B. *et al.* (2013) Powerful sequence similarity search methods and in-depth manual analyses can identify remote homologs in many apparently "Orphan" viral proteins. *Journal of Virology* 88, 10–20

[39] Domazet-Loso, T. and Tautz, D. (2010) Phylostratigraphic tracking of cancer genes suggests a link to the emergence of multicellularity in metazoa. *BMC biology* 8, 66

[40] Šestak, M.S. *et al.* (2013) Phylostratigraphic profiles reveal a deep evolutionary history of the vertebrate head sensory systems. *Frontiers in zoology* 10, 18

[41] Capra, J.A. *et al.* (2013) How old is my gene? *Trends in Genetics* 29, 659–668

[42] Toll-Riera, M. *et al.* (2009) Origin of primate orphan genes: a comparative genomics approach. *Molecular biology and evolution* 26, 603–612

[43] Campbell, M.A. *et al.* (2007) Identification and characterization of lineage-specific genes within the Poaceae. *Plant physiology* 145, 1311–1322

[44] Yomtovian, I. *et al.* (2010) Composition bias and the origin of ORFan genes. *Bioinformatics* 26, 996–999

[45] Sharp, P.M. and Li, W.H. (1987) The codon adaptation index-a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic acids research* 15, 1281–1295

[46] Abrusan, G. (2013) Integration of new genes into cellular networks, and their structural maturation. *Genetics* 195, 1407–1417

[47] Irvine, G.B. *et al.* (2008) Protein aggregation in the brain: the molecular basis for Alzheimer's and Parkinson's diseases. *Molecular medicine* 14, 451

[48] Toll-Riera, M. *et al.* (2011) Role of low-complexity sequences in the formation of novel protein coding sequences. *Molecular biology and evolution* 29, 883–886

[49] Radó-Trilla, N. and Albà, M. (2012) Dissecting the role of low-complexity regions in the evolution of vertebrate proteins. *BMC evolutionary biology* 12, 155

[50] Chen, L. *et al.* (1997) Evolution of antifreeze glycoprotein gene from a trypsinogen gene in Antarctic notothenioid fish. *Proceedings of the National Academy of Sciences* 94, 3811–3816

[51] Ferrada, E. and Wagner, A. (2008) Protein robustness promotes evolutionary innovations on large evolutionary time-scales. *Proceedings of the Royal Society of London B: Biological Sciences* 275, 1595–1602

[52] Capra, J.A. *et al.* (2010) Novel genes exhibit distinct patterns of function acquisition and network integration. *Genome Biol* 11, R127

[53] Naito, K. *et al.* (2009) Unexpected consequences of a sudden and massive transposon amplification on rice gene expression. *Nature* 461, 1130–1134

[54] Lisch, D. (2012) How important are transposons for plant evolution? *Nature Reviews Genetics* 14, 49–61

[55] Teichmann, S.A. and Babu, M.M. (2004) Gene regulatory network growth by duplication. *Nature genetics* 36, 492

[56] Tsai, Z.T.Y. *et al.* (2012) Evolution of cis-regulatory elements in yeast de novo and duplicated new genes. *BMC genomics* 13, 717

[57] Silveira, A.B. *et al.* (2013) Extensive natural epigenetic variation at a de novo originated gene. *PLoS genetics* 9, e1003437

[58] Reinhardt, J.A. *et al.* (2013) De novo ORFs in Drosophila are important to organismal fitness and evolved rapidly from previously non-coding sequences. *PLoS Genetics* 9, e1003860

[59] Chen, S. *et al.* (2010) New genes in Drosophila quickly become essential. *Science* 330, 1682–1685

[60] Chen, W.H. *et al.* (2012) Younger genes are less likely to be essential than older genes, and duplicates are less likely to be essential than singletons of the same age. *Molecular Biology and Evolution* 29, 1703–1706

[61] Luhua, S. *et al.* (2013) Linking genes of unknown function with abiotic stress responses by high-throughput phenotype screening. *Physiologia Plantarum* 148, 322–333

[62] Voolstra, C.R. *et al.* (2011) Rapid evolution of coral proteins responsible for interaction with the environment. *PLoS ONE* 6, e20392

[63] Luhua, S. *et al.* (2008) Enhanced tolerance to oxidative stress in transgenic Arabidopsis plants expressing proteins of unknown function. *PLANT PHYSIOLOGY* 148, 280–292

[64] Gollery, M. *et al.* (2007) POFs: what we don't know can hurt us. *Trends in Plant Science* 12, 492–496

[65] Lacombe, S. *et al.* (2010) Interfamily transfer of a plant pattern-recognition receptor confers broad-spectrum bacterial resistance. *Nature biotechnology* 28, 365

[66] Marshall, E. *et al.* (2011) Cysteine-rich peptides (CRPs) mediate diverse aspects of cell–cell communication in plant reproduction and development. *Journal of experimental botany* 62, 1677–1686

[67] King, G.F. and Hardy, M.C. (2013) Spider-venom peptides: structure, pharmacology, and potential for control of insect pests. *Annual review of entomology* 58, 475–496

[68] de la Vega, R.C.R. *et al.* (2010) Mining on scorpion venom biodiversity. *Toxicon* 56, 1155–1161

[69] de Jonge, R. *et al.* (2011) How filamentous pathogens co-opt plants: the ins and outs of fungal effectors. *Current Opinion in Plant Biology* 14, 400–406

[70] Stergiopoulos, I. and de Wit, P.J. (2009) Fungal effector proteins. *Annual Review of Phytopathology* 47, 233–263

[71] Spanu, P.D. *et al.* (2010) Genome expansion and gene loss in powdery mildew fungi reveal tradeoffs in extreme parasitism. *Science* 330, 1543–1546

[72] Bolton, M.D. *et al.* (2008) The novel *Cladosporium* fulvum lysin motif effector Ecp6 is a virulence factor with orthologues in other fungal species. *Molecular Microbiology* 69, 119–136

[73] Stergiopoulos, I. *et al.* (2010) Tomato Cf resistance proteins mediate recognition of cognate homologous effectors from fungi pathogenic on dicots and monocots. *Proceedings of the National Academy of Sciences* 107, 7610–7615

[74] Elzinga, D.A. and Jander, G. (2013) The role of protein effectors in plant-aphid interactions. *Current opinion in plant biology* 16, 451–456

[75] Tomalova, I. *et al.* (2012) The map-1 gene family in root-knot nematodes, Meloidogyne spp.: A set of taxonomically restricted genes specific to clonal species. *PLoS One* 7, e38656

[76] Mandal, S.M. *et al.* (2013) The attack of the phytopathogens and the trumpet solo: Identification of a novel plant antifungal peptide with distinct fold and disulfide bond pattern. *Biochimie* 95, 1939–1948

[77] Xiao, W. *et al.* (2009) A rice gene of de novo origin negatively regulates pathogen-induced defense response. *PLoS one* 4, e4603

[78] Yadeta, K.A. *et al.* (2014) The Brassicaceae-specific EWR1 gene provides resistance to vascular wilt pathogens. *PLoS one* 9, e88230

[79] Moran, Y. *et al.* (2012) Analysis of soluble protein contents from the nematocysts of a model sea anemone sheds light on venom evolution. *Marine Biotechnology* 15, 329–339

[80] Ovchinnikova, T.V. *et al.* (2006) Aurelin, a novel antimicrobial peptide from jellyfish *Aurelia aurita* with structural features of defensins and channel-blocking toxins. *Biochemical and biophysical research communications* 348, 514–523

[81] Shenkarev, Z.O. *et al.* (2012) Recombinant expression and solution structure of antimicrobial peptide aurelin from jellyfish *Aurelia aurita*. *Biochemical and biophysical research communications* 429, 63–69

[82] Sperstad, S.V. *et al.* (2011) Antimicrobial peptides from marine invertebrates: challenges and perspectives in marine antimicrobial peptide discovery. *Biotechnology advances* 29, 519–530

[83] Haerty, W. *et al.* (2007) Evolution in the fast lane: rapidly evolving sex-related genes in Drosophila. *Genetics* 177, 1321–1335

[84] Li, L. *et al.* (2009) Identification of the novel protein QQS as a component of the starch metabolic network in Arabidopsis leaves. *The Plant Journal* 58, 485–498

[85] Wang, Y. *et al.* (2008) Transcriptome analyses show changes in gene expression to accompany pollen germination and tube growth in Arabidopsis. *Plant physiology* 148, 1201–1211

[86] Piasecka, B. *et al.* (2013) The hourglass and the early conservation models–coexisting patterns of developmental constraints in vertebrates. *PLoS genetics* 9, e1003476

[87] Kalinka, A.T. *et al.* (2010) Gene expression divergence recapitulates the developmental hourglass model. *Nature* 468, 811–814

[88] Quint, M. *et al.* (2012) A transcriptomic hourglass in plant embryogenesis. *Nature* 490, 98–101

[89] Knoll, A.H. (2011) The multiple origins of complex multicellularity. *Annual Review of Earth and Planetary Sciences* 39, 217–239

[90] Kass-Simon, G. and Scappaticci, Jr., A.A. (2002) The behavioral and developmental physiology of nematocysts. *Canadian Journal of Zoology* 80, 1772–1794

[91] Hwang, J.S. *et al.* (2007) The evolutionary emergence of cell type-specific genes inferred from the gene expression analysis of Hydra. *Proceedings of the National Academy of Sciences* 104, 14735–14740

[92] Shigenobu, S. and Stern, D.L. (2012) Aphids evolved novel secreted proteins for symbiosis with bacterial endosymbiont. *Proceedings of the Royal Society B: Biological Sciences* 280, 20121952–20121952

[93] Newman, A.M. and Cooper, J.B. (2011) Global analysis of proline-rich tandem repeat proteins reveals broad phylogenetic diversity in plant secretomes. *PLoS One* 6, e23167

[94] Beilstein, M.A. *et al.* (2010) Dated molecular phylogenies indicate a Miocene origin for *Arabidopsis thaliana*. *Proceedings of the National Academy of Sciences* 107, 18724–18728

[95] Wang, H. *et al.* (2009) Rosid radiation and the rapid rise of angiosperm-dominated forests. *Proceedings of the National Academy of Sciences* 106, 3853–3858

[96] Moore, M.J. *et al.* (2007) Using plastid genome-scale data to resolve enigmatic relationships among basal angiosperms. *Proceedings of the National Academy of Sciences* 104, 19363–19368

[97] Schneider, H. *et al.* (2004) Ferns diversified in the shadow of angiosperms. *Nature* 428, 553

[98] Soltis, P.S. *et al.* (2002) Rate heterogeneity among lineages of tracheophytes: integration of molecular and fossil data and evidence for molecular living fossils. *Proceedings of the National Academy of Sciences* 99, 4430–4435

[99] Hedges, S.B. *et al.* (2004) A molecular timescale of eukaryote evolution and the rise of complex multicellular life. *BMC evolutionary biology* 4, 2

[100] Hedges, S.B. *et al.* (2001) A genomic timescale for the origin of eukaryotes. *BMC Evolutionary Biology* 1, 4

[101] Noffke, N. *et al.* (2013) Microbially induced sedimentary structures recording an ancient ecosystem in the ca. 3.48 billion-year-old Dresser Formation, Pilbara, Western Australia. *Astrobiology* 13, 1103–1124

[102] Weng, J.K. *et al.* (2012) The rise of chemodiversity in plants. *Science* 336, 1667–1670

[103] Ngaki, M.N. *et al.* (2012) Evolution of the chalcone-isomerase fold from fatty-acid binding to stereospecific catalysis. *Nature* 485, 530

[104] Balandrin, M.F. *et al.* (1985) Natural plant chemicals: Sources of industrial and medicinal materials. *Science* 228, 1154–1160

[105] Oliver, D.J. *et al.* (2009) Acetyl-CoA–life at the metabolic nexus. *Plant science* 176, 597–601

[106] Block, A. *et al.* (2014) The origin and biosynthesis of the benzenoid moiety of ubiquinone (coenzyme Q) in Arabidopsis. *The Plant Cell* 26, 1938–1948

[107] Pichersky, E. and Gang, D.R. (2000) Genetics and biochemistry of secondary metabolites in plants: An evolutionary perspective. *Trends in plant science* 5, 439–445

[108] Xue, W. *et al.* (2013) The investment in scent: time-resolved metabolic processes in developing volatile-producing *Nigella sativa L.* seeds. *PLoS one* 8, e73061

[109] Fraser, C.M. and Chapple, C. (2011) The phenylpropanoid pathway in Arabidopsis. *The Arabidopsis Book* p. e0152

[110] Weigel, D. and Mott, R. (2009) The 1001 genomes project for *Arabidopsis thaliana*. *Genome Biol* 10, 107

[111] Long, Q. *et al.* (2013) Massive genomic variation and strong selection in *Arabidopsis thaliana* lines from Sweden. *Nature Genetics* 45, 884–890

[112] Gan, X. *et al.* (2011) Multiple reference genomes and transcriptomes for *Arabidopsis thaliana*. *Nature* 477, 419–423

CHAPTER 3. `phylostratr`: A FRAMEWORK FOR PHYLOSTRATIGRAPHY

Zebulun Arendsee, Jing Li, Urminder Singh, Arun Seetharam, Karin Dorman, Eve Syrkin Wurtele

Modified from a paper published in *Bioinformatics*

Abstract

Motivation: The goal of phylostratigraphy is to infer the evolutionary origin of each gene in an organism. This is done by searching for homologs within increasingly broad clades. The deepest clade that contains a homolog of the protein(s) encoded by a gene is that gene's phylostratum. **Results:** We have created a general R-based framework, `phylostratr`, to estimate the phylostratum of every gene in a species. The program fully automates analysis: selecting species for balanced representation, retrieving sequences, building databases, inferring phylostrata, and returning diagnostics. Key diagnostics include: detection of genes with inferred homologs in old clades, but not intermediate ones; proteome quality assessments; false-positive diagnostics, and checks for missing organellar genomes. `phylostratr` allows extensive customization and systematic comparisons of the influence of analysis parameters or genomes on phylostrata inference. A user may: modify the automatically-generated clade tree or use their own tree; provide custom sequences in place of those automatically retrieved from UniProt; replace BLAST with an alternative algorithm; or tailor the method and sensitivity of the homology inference classifier. We show the utility of `phylostratr` through case studies in *Arabidopsis thaliana* and *Saccharomyces cerevisiae*.

3.1 Introduction

The diversity of species on earth rises as species adapt to changing environments through the continual modification and re-purposing of existing genes and the emergence of new genes. A cornerstone of evolution is that genes (often after duplication) continually evolve new functions by modification of existing components and acquisition of new functional motifs [1, 2]. More recently, it has become apparent that a second major mechanism by which organisms gain diversity is the *de novo* birth of genes [3, 4, 5, 6, 7]. Phylostratigraphy is the process of determining the phylogenetic origin of every gene in a genome [3]. Phylostratigraphy classifies each gene by its age, where age is defined relative to the branch of the phylogenetic tree on which the gene is inferred to have first appeared. Thus the age of a gene, in the phylostratigraphy context, is not the time since the latest gene duplication event, but rather the time since the original ancestor of the gene family appeared. Phylostratigraphy is premised on the theory that new genes arise continuously. Newly-emerged genes (termed "orphans" when they are *species-specific* [5], include both genes encoding unique, *de novo*-emerged proteins that arose from non-genic sequence, and genes encoding unique *de novo*-emerged proteins that arose from within an existing gene (e.g., from 5' or 3' non-coding sequence, unspliced introns, or a change of reading frame) [6]. A second type of orphan would result from the ultra-rapid evolution of an existing protein such that it can not be recognized in a related species.

Phylostratigraphy has been applied to infer the novel proteins that support evolutionary emergence of new structures and functions. For example, the over-represented expression of human genes from the chordate phylostratum in the midbrain has been used to infer the genes involved in the origin of this structure [8]. Also, the unique expression of an orphan (species-specific) gene in the shoot apical meristem of *Arabidopsis* has led to conjecture on its role in this structure [9]. By linking genes to their point of origin, we can identify candidate genes for involvement in clade-specific pathways [10].

Speciation is thought to be enabled in part by emergence of new genes with novel functions that permit organisms to respond to changing environments [10, 5, 6]. Phylostratigraphy provides a first step towards understanding the evolutionary paths a protein-coding gene may follow as it evolves from non-genic sequence or as a completely novel reading frame within an existing gene [6]. The changes that a young gene has undergone can be inferred by analysis of all the members of a lineage-restricted family. For example, Antarctic notothenioid fish gained an anti-freeze protein, not observed in other lineages, through the tandem duplication of a glycotripeptide [11]. The very early stages in evolution of a new protein-coding gene can be revealed by comparing the sequences of that gene across distinct populations within a species ([12]

While phylostratigraphy may appear conceptually simple, in practice it is challenging to infer phylostratum assignments. Standard phylostratigraphic classification depends solely on the single most distant inferred homolog [3]. This makes phylostratigraphy sensitive to false positives, such as homologs that are inferred due to repetitive sequences. Orphan genes are likely to be unannotated or mis-annotated. Because by definition "orphan proteins" have no homologs in other species, recognizing them in genomes and distinguishing them from noise is a major challenge [13, 7]. Sequence data may be incomplete or inaccurate and organelle proteins and genomes may be missing. Phylogenetic trees may be incorrect or ambiguous (i.e., there may be many unresolved nodes). The data can be systematically biased: some phylostrata may be over-represented in a given analysis and approaches to gene annotation may differ across time and by species. Finally, there are the mundane but time-consuming complexities of implementing the pipeline, such as acquiring sequence data, building the databases, and managing the proteome files and their phylogenetic relationships. These factors, which can introduce errors in phylostratigraphic designations that cannot be easily caught, highlight the importance of using automation and diagnostics in phylostratigraphic analysis.

A major technical challenge to phylostratigraphy is the difficulty of distinguishing short and/or rapidly evolving genes from genes that have arisen *de novo* [14, 15, 16, 17, 18, 19]. There is a current controversy over the extent of bias this introduces to phylostratigraphy [16, 17, 20]. If sequence data from recently-diverged subspecies can be incorporated into the analysis, distinguishing rapidly evolving genes from *de novo* genes may be feasible.

A more fundamental challenge to phylostratigraphy is that gene "age" is not always well-defined. For example, a protein may be comprised of multiple domains that are gained and lost over time. Thus each domain may have a different age. In phylostratigraphy, the "age" of such proteins is designated as the age of its oldest domain, even if this oldest domain was a late insertion into the younger gene. To understand proteins on a domain basis, the protein domains can be used as the search unit for phylostratigraphy [21].

There are currently no phylostratigraphic pipelines that offer a high-degree of automation, customization, and diagnostics. A web-based tool for phylostratigraphy, ORFanFinder [22], limits analysis to the strata corresponding to the named ranks of the NCBI common tree, which is too low-resolution for many purposes, and does not provide diagnostics, statistics, or customization. There are published scripts available for phylostratigraphy but these are tailored to the needs of the specific study (e.g., [23, 24, 25]). The MetaPhOrs web tool has searchable, precalculated orthology and paralogy predictions for pairs of species based on phylostratigraphy inference [26].

Here, we describe `phylostratr`, a customizable R package for reproducible, whole-genome phylostratigraphy. `phylostratr` automates what can be automated (e.g., sequence retrieval, database building, file handling, and NCBI tree building) and makes customizable what may need to be customized (e.g., input clade trees, homology classifiers, and protein sequence data). The package provides a suite of tools for diagnosing common pitfalls, catching phylostratigraphic oddities, and preparing output data for

specialized analysis. We envision `phylostratr` being used as a flexible pipeline for phylostratigraphic annotation of genomes and as a framework for downstream analysis of the emergence and development of genes.

3.2 Implementation

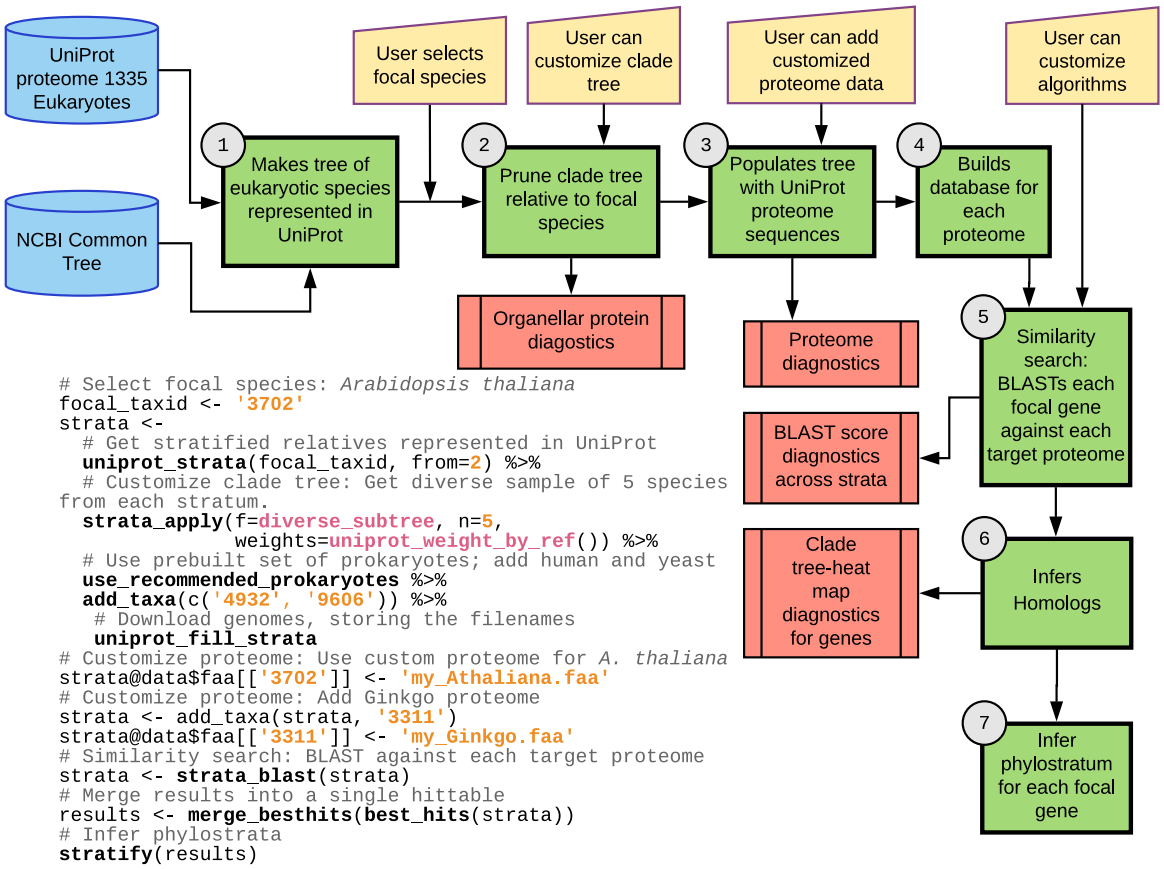
The purpose of `phylostratr` is to make phylostratigraphic analysis accessible, introspective, reproducible, and flexible. We make the pipeline *accessible* by automating much of the complex data handling (e.g., tree operations and proteome sequence retrieval) and sequence database tracking. `phylostratr` is *introspective* in that it provides powerful tools for finding flaws and irregularities in both inputs and results. `phylostratr` pipelines are *reproducible* since the entire pipeline, from data retrieval to diagnostics to downstream analysis, can be written and shared as a single R script. *Flexibility* is achieved by implementing the tools as a library within the powerful R environment allowing the seven core functions in `phylostratr` to be easily customized.

In the following subsections we walk through these seven core functions of `phylostratr` as outlined in **Figure 3.1**.

3.2.1 Create clade tree

Core Function 1 creates a large clade tree by mapping all species in the UniProt Proteome database [27] to the NCBI common tree [28]. The UniProt Proteome database includes only species with sequenced genomes.

Our choice to limit the search space to the proteomes of a fixed number of species is somewhat unusual. It is common practice in phylostratigraphy to search for homologs in the largest database possible (usually the NCBI `nr` database). What species are involved, and whether their proteomes are complete, is often not considered. *Tyrannosaurus rex*, as an extreme example, is represented by a single collagen sequence [29]. We advocate using well-annotated proteomes from species with well-sequenced genomes or transcriptomes,




```
# Select focal species: Arabidopsis thaliana
focal_taxid <- '3702'
strata <-
  # Get stratified relatives represented in UniProt
  uniprot_strata(focal_taxid, from=2) %>%
  # Customize clade tree: Get diverse sample of 5 species
from each stratum.
  strata_apply(f=diverse_subtree, n=5,
               weights=uniprot_weight_by_ref()) %>%
  # Use prebuilt set of prokaryotes; add human and yeast
  use_recommended_prokaryotes %>%
  add_taxa(c('4932', '9606')) %>%
   # Download genomes, storing the filenames
   uniprot_fill_strata
# Customize proteome: Use custom proteome for A. thaliana
strata@data$faa[['3702']] <- 'my_Athaliana.faa'
# Customize proteome: Add Ginkgo proteome
strata <- add_taxa(strata, '3311')
strata@data$faa[['3311']] <- 'my_Ginkgo.faa'
# Similarity search: BLAST against each target proteome
strata <- strata_blast(strata)
# Merge results into a single hittable
results <- merge_besthits(best_hits(strata))
# Infer phylostrata
stratify(results)
```

Figure 3.1 Overview of `phylostratr`. **Flow chart:** `phylostratr` 1) creates a clade tree from the species currently represented in UniProt and the NCBI tree of life; 2) trims this tree, using an algorithm designed to maximize the evolutionary diversity of species; 3) retrieves proteomes from UniProt Proteome [27]; 4) builds a database for each proteome; 5) performs a pairwise BLAST of the focal species proteome against the proteome of each species in the tree; 6) finds the "best hits" for each focal gene against each target species; 7) based on the best hits, assigns each gene to the phylostratum associated with the deepest clade to which the gene has an inferred homolog. The focal species (3702) is the only input required to run the default analysis. **Green boxes**: R core phylostratigraphy functions. **Yellow trapezoids**: customizable features. **Salmon boxes**: final outputs and diagnostics. **R code, customized for an *A. thaliana* study:** The maximum number of species to be included in each clade are specified ($n = 5$), the actual number may be lower if there are not five representatives. The user may add weights to the species-selection algorithm, here we set a preference for UniProt reference species; two eukaryotic species (humans and yeast) and and 85 prokaryotic species were added to the tree. The default UniProt proteome for the focal species was replaced with a custom one 'my_thaliana.faa'. We added the Ginkgo proteome to offset sparse coverage of the Spermatophyte stratum.

because it enables more accurate inference of the species that do *not* have a homolog, not just which species *do* have one. Capturing this information is difficult using only the `nr` database, since the quality of the proteomes for each species is not usually evident. If no hit is found against species X, one cannot easily determine whether: 1) too little sequence is available for X (e.g. only the organelle proteins are included); or 2) X actually has no homolog. To minimize this problem, we base our analysis on proteomes that should be reasonably complete, specifically the UniProt Proteome collection. This is, however, only the `phylostratr` default, and the user is free to use sequence data outside UniProt.

3.2.2 Prune clade tree

Core Function 2 prunes the clade tree such that it retains a phylogenetically diverse selection of representatives for each phylostratum. The algorithm is outlined in **Figure 3.2**.

3.2.2.1 Leaf selection algorithm

Considering the focal species, the node toward the root is the parent. The other children of the parent are the 0th cousins (siblings). If a focal gene has no homolog among the 0th cousins, then it is an orphan gene. Similarly, 1st cousins are species that share a common grandparent and so one until Nth cousins that share the last universal common ancestor of all life. Thus the set of $(i - N)$th cousins are the representative species for the ith phylostratum.

Core Function 2 is designed to provide a balanced representation of species for each phylostatum. To this end, each set of cousins in the species tree is independently pruned by the leaf selection algorithm (see **Figure 3.2**). This algorithm selects up to a given number of species per cousin set and ensures the selected species are as distantly related as possible while respecting a weight vector indicating the preference for each species. These pruned sets of cousins are then reassembled into the final tree.

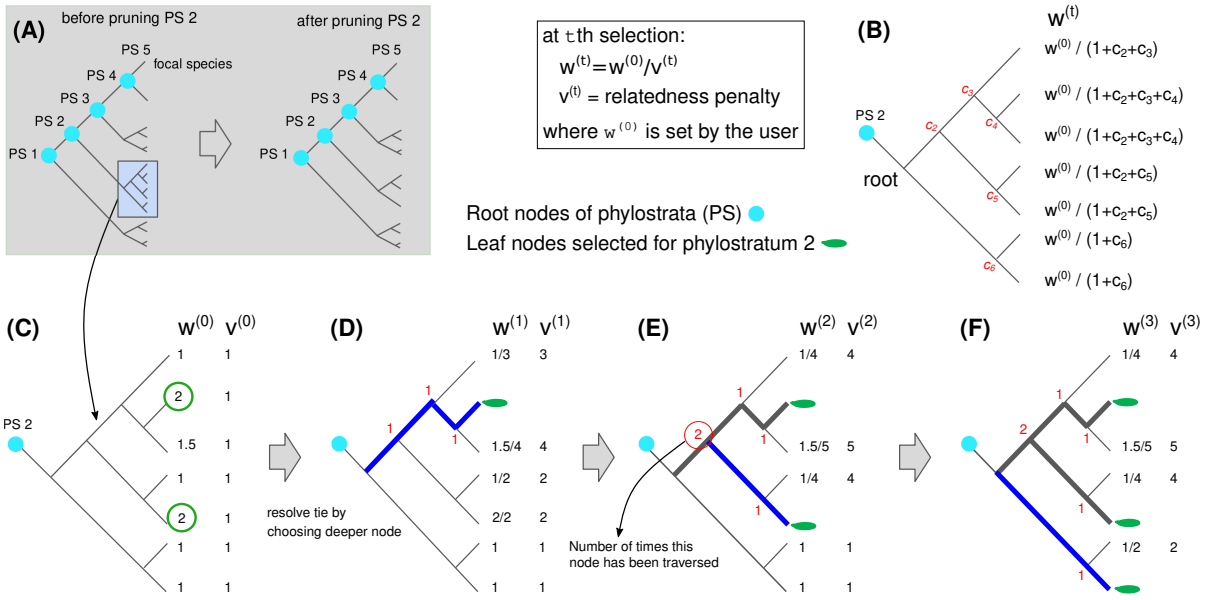


Figure 3.2 Pruning a clade tree to optimize evolutionary diversity. Each leaf represent a species. **A.** The full clade tree after pruning PS 1, but before pruning PS 2-4 (left) and the same tree after pruning PS 2 (right). **B.** Generic representation of selection of a leaf from a phylostratum (PS2). The user provides an initial weight vector $\vec{w}^{(0)}$ for each leaf; in this case the weights are chosen based on the quality of proteome, with species with better proteomes available having higher weights. The initial relatedness coefficients are all set as equal to 1, indicating no preference for any specific branch. Once the first leaf is chosen, selection of subsequent leaves is penalized according to the number of shared branches. c_3 refers to the number of leaves that have been selected that share node 3. **C-F.**. The process of pruning PS2. PS2 tree at initiation of pruning (C). Leaves with the greatest $\vec{w}$ are selected and the process recurses (D-F). The red numbers indicate the number of times a given node has been transversed. Equal weights (e.g., green circles in C) are resolved first by node depth and then alphabetically by species name.

Within each phylostratum, the `phylostratr` leaf selection algorithm (the recursive algorithm represented by **Equation 3.1** - **Equation 3.3**) chooses a diverse set of leaves (species) while respecting a weight vector $\vec{w}^{(0)}$ provided by the user. The algorithm is outlined in **Figure 3.2**, where Panel B is an overview and Panels C-F provide an example of the selection process. The user may assign weights to reflect the quality of proteomes of the individual species, and possibly to include other user-designated criteria, such as giving a low weight to a species that is a parasite and would have a very reduced genome.

The algorithm begins by selecting the leaf with the highest relatedness-adjusted weight

$$w_k^{(t)} = \frac{w_k^{(0)}}{v_k^{(t)} + 1} \tag{3.1}$$

where $w_k^{(t)}$ is the relatedness-adjusted weight for the kth leaf node after the tth selection, ${w_k}^{(0)}$ is the initial weight of the kth leaf node, and $v_k^{(t)}$ is the relatedness penalty for leaf k. If the user does not provide a weight vector, the weights default to 1.

The relatedness penalty for a given (candidate) leaf node, k, is the total number of times its ancestral internal nodes are shared with currently selected leaves. Thus, the relatedness penalty for a leaf is greatest when it has the most nodes in common with those of other selected leaves. It is calculated as follows.

Let $c_i^{(t)}$ be the number of times that internal node i is ancestral to all currently selected leaf nodes. At the initial leaf selection run, $c_i^{(0)} = 0$ for all nodes i. At each successive run, after a new leaf is selected, the c_is for each internal node are recursively updated in a post-order tree traversal using the equation

$$c_i^{(t)} = \sum_{j \in \mathcal{D}_i} c_j^{(t)}, \tag{3.2}$$

where $\mathcal{D}_i$ is the set of daughter nodes of node i. The relatedness penalty for each leaf node is then calculated as $v_k^{(t)} = v_{i,0}^{(t)} + 1$ after another post-order tree traversal given by recurrence equation

$$v_{i,l-1}^{(t)} = v_{i,l}^{(t)} + c_i^{(t)}, \text{with } l = d, d-1, \ldots, 1 \tag{3.3}$$

initialized with $v_{i,d}^{(t)} = 0$, where d is the depth (in nodes) of node i.

After calculating $\vec{w}^{(t)}$ for all leaf nodes, we select the leaf node k with the highest $w_k^{(t)}$. Ties are resolved by choosing the leaf node with the greatest number of internal nodes between it and the root. Any remaining ties are resolved alphabetically. After selecting leaf node k, then $c_k^{(t)} = 1$ and we continue to the next iteration (**Equation 3.1**). The algorithm continues selecting leaves until the user-set threshold number of species is reached or until there are no remaining species to be selected (Panel F, **Figure 3.2**). At that point, the phylostratum is fully pruned.

Assuming each phylostratum is an ultrametric phylogenetic tree, i.e., a tree in which all leaf nodes are the same evolutionary distance from the root node, the algorithm maximizes the total evolutionary distance traversed by the species it selects, and reduces the selection of very closely related species. Thus, the species that are selected will be a diverse representation of the phylostratum. Consequently, this method is unlikely to obtain a false negative for homology by accidentally considering only closely-related species that have all lost a gene that existed in a common ancestor.

3.2.2.2 User modifications

The user may refine the pruning strategy at any step from creating the clade tree to modifying the selection algorithm. The NCBI common tree may be replaced (all or in part) with a custom tree; this is particularly important in the resolution of polytomies (unresolved lineages with multiple branches from a single node), which are common in the NCBI tree (**Figure S5-7**). Species and subspecies (e.g., ecotypes, races, lineages, varieties) can be represented as leaves in the tree as long as there is an associated proteome. Proteomes from subspecies relatives of the focal species can be added to resolve very recent genomic events such as emergence of orphan genes. The leaf selection step can be modified by altering the weights in the selection algorithm, for example, to favor inclusion or exclusion of particular species or to set a preference for species with

high quality proteomes (see **Figure 3.1** where we set a preference for UniProt reference proteomes).

The leaf selection algorithm can be extended to non-ultrametric trees if the evolutionary distances of branches are known, simply by computing branch length-weighted counts of selected species in **Equation 3.2** of the algorithm. The user may omit the leaf selection step entirely, in which case all proteomes in UniProt would be included as leaves in the analysis.

3.2.2.3 Motivations for pruning and diversifying the tree

Phylostratigraphy is highly sensitive to false positives, since any phylostratigraphic classification of a gene depends solely on the single most distant inferred homolog. As the number of species evaluated in a phylostratigraphic analysis increases, several things may occur [16, 15, 18, 30]. 1) The chance of false negatives (e.g., caused by proteomes representing a phylostratum lacking the homolog by chance deletion or incomplete annotation) is decreased, assuming there are sufficient representatives in each clade. 2) The standard for statistically significantly homology is increased (since the analysis adjusts for the total number of species). If the species that are added are highly similar to species already included in the study they will add little new information; in this case, any increase in significance threshold associated with adding more species would result in a bias towards younger phylostratigraphic classifications. 3) The number of false positives increases, due to convergent evolution, horizontal gene transfer, and repeats.

`phylostratr` gives the user fine control of the selection and number of species representing each phylostatrum. Thus the automatic data handling of phylostratr facilitates testing of database size effects in actual genome data and enables evaluation of the customizable pruning/diversification feature we have implemented (Module 2 of **Figure 3.1**).

We developed a species pruning/diversification feature in `phylostratr` for several reasons.

The first motivation for pruning with selection for diverse species is to reduce bias caused by the non-random evolutionary distribution of species with available proteomes. One clade might have two available proteomes while another might have five thousand.

A second motivation for pruning is to decrease bias due to the *dependences* among proteomes of related organisms. In adjusting for multiple testing, BLAST makes the assumption that the homology scores between hits of related species are independent [31], yet, because closely related species inherit similar genes, there are massive deviations from this assumption. A pruned tree, i.e, a tree with fewer and more diverse species, reduces the reliance on this implicit assumption that genes are independent between target species. For instance, in trying to determine whether a given plant gene is represented among non-plant Eukaryotes, searching the chimpanzee, baboon, and orangutan proteomes after already having searched the human proteome will yield little additional information, since these proteomes are so similar. In this case, adding many closely-related representatives of one phylostratum raises the E-value (expected number of random hits with score greater than or equal to the observed score) while adding little extra information; this shifts all phylostratal classifications towards *younger* assignments. The algorithm we have implemented reduces this bias, and the false positives it causes, by pruning the tree to provide a balanced representation across strata and diversity within strata.

A third motivation for pruning is to reduce the number of false positives that are caused by rare random events that are not modeled by the homolog detection method. Homolog detection algorithms (**Section 3.2.5**) account for the chance events under a simplistic model of sequence evolution; they do not account for convergent evolution (for example as a consequence of natural selection for functional motifs or neutral expansion of repeats). Thus, as the number of representative species for a stratum grows, the

chance of obtaining false positives also rises. As a result, homology inference is biased towards highly sampled strata. Our strategy for pruning the tree reduces this bias.

A final motivation for pruning is that it reduces computational time and simplifies downstream analysis. In particular, limiting the number of proteomes speeds up the similarity search (**Section 3.2.5**).

3.2.3 Acquire proteome sequences

Core Function 3 automatically downloads proteomes for species that are represented in the UniProt database. Alternatively, the user may input their own sequence data, or can add proteomes in addition to, or in place of, the UniProt data. For example, in our *A. thaliana* analysis we added the Ginkgo proteome to ensure we had at least one reasonable representative of the spermatophyte clade (**Figure 3.1**).

An important modification would be to extract the translated ORFs from the genome or from a transcriptome (e.g., using the `findorf` program [32]) and use these as the proteome input. This could mitigate false results due to use of proteomes from badly annotated genomes. The downside is that even the small yeast genome has over 30,000 ORFs >150 nt, and *A. thaliana* has over 2 million ORFs; thus supplying proteomes of translated ORFS would increase the search space and thereby decrease the sensitivity.

3.2.4 Build proteome databases

Core Function 4 builds databases from the proteome sequences in order to perform the similarity search. There are three possible ways to partition the sequences into databases. 1) The sequences could be placed in a single pooled database; 2) sequences could be partitioned into one database per phylostratum; or, 3) each species could have its own database. The partitioning approach determines how the E-value (the expected number of hits with greater or equal score) is interpreted. The first way, a pooled database, is by far the most common in the literature. However, we chose the third option — each species having its own database — since this permits interpretation of E-values for each

species without a statistical dependence on other species in the analysis. This allows species to be added or removed from the study without affecting previously computed E-values. New species may be then incorporated as proteome sequences become available, without having to redo the whole analysis.

3.2.5 Similarity search

Core Function 5 searches for similarity between the proteins encoded in the focal species and all proteins from all target species in the study. BLAST is the most popular similarity search algorithm, and is the default tool for phylostratigraphy [17]. `phylostratr` offers dedicated handling for BLAST: it automatically builds BLAST databases for all proteomes, runs BLAST alignment, and merges the results. Alternatively, the user may opt to run the BLAST alignment search on a distributed system and then feed the results back into `phylostratr`.

3.2.6 Infer homologs

Core Function 6 infers homology from the similarity results. In most phylostratigraphic studies, homology detection entails selecting an E-value sequence similarity threshold for a BLAST search of query proteins against a pooled database of available protein data for all species in all strata. All hits to the query protein with E-values below the cutoff are recorded and the significant hit in the most phylogenetically distant species determines the phylostratum classification. An E-value below the significance cutoff implies a homolog in the ancestor. The minimum E-value can be interpreted as a p-value under appropriate assumptions [33], in which case this statistical procedure is equivalent to testing the null hypothesis that there are no homologous proteins anywhere in the phylogenetic tree.

To account for the total number of species in the study, we multiply the E-values obtained by searching each focal species against the BLAST database for each target species by the total number of target species in the study. This will yield adjusted E-

values similar to those obtained by searching a pooled BLAST database. This is true since a BLAST E-value is directly proportional to database size — doubling the size of the database doubles the E-value for every hit against the database. Assuming all species have proteomes of roughly the same size, every query searched against a pooled database of n species will have an E-value roughly equal to ne, where e is the E-value of the query searched against just one species. Like a BLAST against the pooled database, this hypothesis testing procedure controls the chance of detecting a false homolog anywhere in the tree.

Each algorithm for homology prediction has its strengths and weaknesses (systematically assessed in [25]). `phylostratr` has flexibility to add/use different homology inference algorithms such as similarity coverage, or domain prediction (reviewed in [25]); for example, HMMER [34] could replace BLAST throughout the analysis. This would enable a user to compare methods across clades or supply several types of homology evidence.

3.2.7 Infer phylostrata

Core Function 7 infers the phylostratrum of each focal species gene given the homology results. In conventional phylostratigraphy, and in `phylostratr` as currently implemented, each gene is assigned to the most evolutionarily distant phylostratum that contains an inferred homolog.

3.2.8 Access and diagnostics

After executing the preceding 7 core functions, the `phylostratr` user may access and analyze their results through a rich suite of diagnostic and exploratory tools provided by `phylostratr` (**Figure 3.1**). Diagnostics include protein quality assessment (**Table S1**), organelle genome checking (**Table 3.2**), heat maps of inference methods and of skipped clades (**Figure 3.3**), and gene by clade tree-heat maps (**Figure 3.4**). For example, the

Table 3.1 A comparison of the partitioning of genes between phylostrata in the current study and that of [6]. Since the two studies used different annotations of the *A. thaliana* genome (TAIR10 and Araport11), only the genes present in both annotations are counted.

Phylostratum	This study	(Arendsee, 2014)
Cellular organisms	10624	8936
Eukaryota	6001	7478
Viridiplantae	1498	1899
Embryophyta	3952	3365
Tracheophyta	260	463
Spermatophyta	1341	1226
Mesangiospermae	541	867
Eudicotyledons	367	246
Pentapetalae	197	396
Rosids	125	177
Malvids	79	46
Brassicaceae	963	976
Camelineae	99	97
Arabidopsis	180	175
Arabidopsis thaliana	692	572

effects of different p-value cutoffs on phylostrata assignments is visualized for yeast in **Figure 3.3A** and Arabidopsis in **Figure 3.3B**.

3.3 Results and Discussion

There are three main advantages of using `phylostratr` for a phylostratigraphic analysis. First, using `phylostratr` removes many time-consuming manual steps and reduces the possibilities of errors. Second, `phylostratr` provides detailed diagnostic plots, which can be valuable for the original researcher or for a researcher who is comparing their own results to an existing study. Third, a `phylostratr` study is reproducible, simple to modify using new protein data or new algorithms, and the results can easily be incorporated into downstream R analyses.

We demonstrate the use of `phylostratr` for phylostratigraphic analyses in two test case studies: *Arabidopsis thaliana* and *Saccharomyces cerevisiae*. In both case studies, to

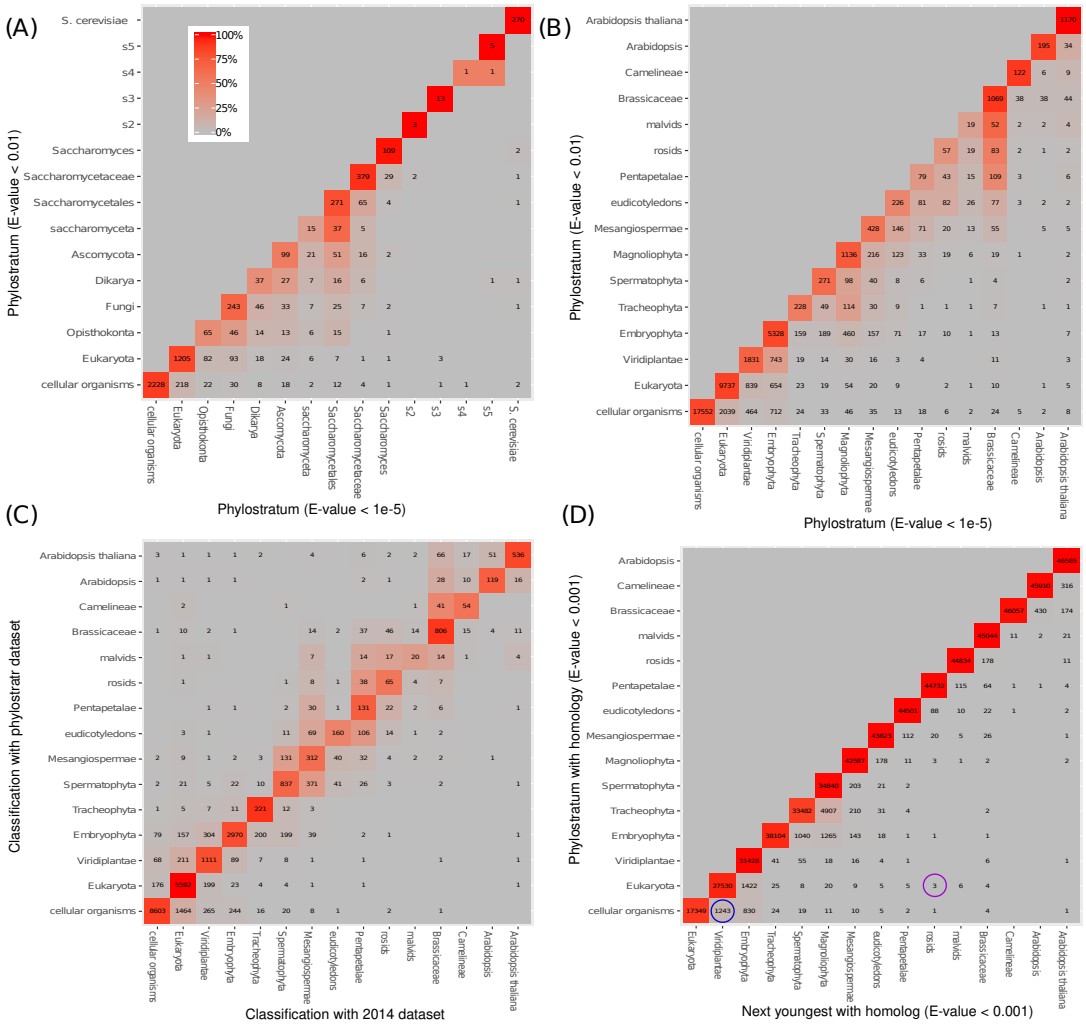


Figure 3.3 Comparisons of phylostratigraphic analyses. **(A)** *S. cerevisiae*. Phylostrata S2-S5 are sub-genus clades of species closely related to *S. cerevisiae* (see **Figure S4**) **(B-D)** *A. thaliana*. **(A,B)** Changing from a high adjusted P–value cutoff (0.05) to a lower cutoff (10^{-5}) results in a shift towards younger classifications. **(C)** `phylostratr` results versus [6]. The largest difference between the studies is the 1688 more genes assigned to the cellular organisms stratum by `phylostratr`, likely because of a more diverse representation of prokaryotic species in the current study (85 vs 12 species). **(D)** Skipped clades: the numbers of genes that have an inferred homolog in a phylostratum (Y axis) and their next closest inferred homolog in a younger phylostratum (X axis). Genes far from the diagonal are likely false positives or potentially represent a horizontal gene transfer. For example, three of the genes with a homolog in the Eukaryota stratum have no other inferred homolog until malvids (**purple circle**). The 1243 genes present in cellular organisms but that then skip to viridiplantae may reflect the horizontal transfer during endosymbiosis (**blue circle**).

illustrate diagnostic features of `phylostratr`, we intentionally retained a few species with low quality nuclear or organelle gene annotations. The full R scripts and diagnostic output for the *A. thaliana* and *S. cerevisiae* case studies are included in the Supplementary Information.

Ultimately, identifying and avoiding false positive and false negative assignments depends not only on methodology, but also on the technical aspects of genome assembly and annotation. As genome quality improves, `phylostratr` will make it possible to efficiently compare different target genome versions.

3.3.1 Phylogenetic tree, data collection

For the *A. thaliana* case study, we used the default NCBI common tree and UniProt proteomes for the target search space. We set a preference for reference proteomes (weight of 1.1) and designated a maximum of five representatives per phylostratum using the algorithm explained in **Section 3.2.2.1**. From the 1335 eukaryotic proteomes retrieved by UniProt, we filtered out a diverse set of 45 proteomes. We then added human and *S. cerevesiae* genomes, because they are highly curated. After looking at the diagnostic plots from an initial run, we noted that the genome of the single representation of the Spermatophyta stratum in NCBI, *Picea glauca*, was sparsely annotated. To obviate this under-annotation, we added the proteome translated from the Spermatophyte *Ginkgo biloba* from the 1KP project [35, 36, 37, 38] to the target search space. The final code is shown in **Figure 3.1**.

The *S. cerevisiae* case study was similar to that of *A. thaliana*, except that the NCBI common tree genus node *Saccharomyces* was the immediate parent of many species, thus to resolve the differences between these species, we replaced this branch with our own customized clade tree (**Figure S4**).

For both case studies, we selected a diverse set of 85 prokaryotic species, comprised of one species from each class of Bacteria and Archaea, sampled at random from the

UniProt reference proteomes. Despite this sampling, the diversity of the prokaryotes genomes is so great, their content so fluid, and their sequence so highly divergent from eukaryotic species, that some genes with a prokaryotic origin will likely be missed.

3.3.2 `phylostratr` comparison to published results

We compared the `phylostratr` results for *A. thaliana* to those from an earlier study in 2014 [6]. To account for the annotation differences between the 2014 study, which used the TAIR10 annotation of *A. thaliana* made in 2011, and the current study, which uses the Araport11 annotations made in 2017 [39], we compared only the gene models common to both annotations (see **Table 3.1**). The 2014 study used a manually-curated selection of representative species whereas the current study uses automatically-selected UniProt proteomes, supplemented with the *Ginkgo* proteome. Because of this, the strata differed slightly between the studies, and we compared only the common strata. For example, the Brassicales clade was represented in the 2014 study but not the current study, so for our comparison, those proteins classified as Brassicales in the 2014 study were merged into the Brassicaceae clade.

The phylostratigraphic classifications were similar in both studies (**Table 3.1**), with more genes assigned to a deeper origin in the current study. A more in-depth comparison, shown in the `phylostratr` diagnostic of **Figure 3.3C**, maps how gene assignments changed between the studies. This figure indicates that the differences between studies seen in (**Table 3.1**) are due mostly to transitions of assignments to closely-related phylostrata. The differences are likely due mostly to two factors: the expansion of the number of species included in the `phylostratr` study (e.g., 85 prokaryotes in the current study versus 12 in the 2014 study) and the improved sequence data and the increased number and quality of genome annotations in 2018 compared to 2014. Thus, several thousand more genes were traced back to an earlier origin by the `phylostratr` analysis.

Table 3.2 `phylostratr` diagnostics: mitochondrial, chloroplast and total proteins in UniProt for each species in the four youngest phylostrata of the *Arabidopsis thaliana* study. See **Table S2** for full list.

Species	Phylostratum	Mitochondrial	Chloroplast	Total
Arabidopsis lyrata	Arabidopsis	1	101	33099
Capsella rubella	Camelineae	0	92	29000
Brassica napus	Brassicaceae	153	54	62632
Brassica oleracea	Brassicaceae	0	0	58540

3.3.3 Protein and organelle diagnostics

`phylostratr` provides functions that summarize and visualize proteome statistics. These can be used as a quick way to identify irregularities in proteome qualities. For example, the reference proteome of the primate *Pan paniscus* (a bonobo) was annotated as having only 802 genes (**Table S7**). The median protein length of 181 amino acids reported for *P. paniscus* was also far shorter that of the other primate species (human). This suggests the genome was incomplete, poorly sequenced, or poorly annotated and should not be included in the study.

`phylostratr` can determine which proteins in the UniProt proteome database are encoded by genes in organelle genomes, particularly important because organellar genomes are often missing. If the focal species contains organelle genomes, but the target species do not, the organelle genes can appear to be younger than they actually are. Further compounding this problem of the paucity of reported organellar proteomes is the continual flux of genes from plastid genome to nuclear genome over evolutionary time. In the *A. thaliana* study, 22 of the 41 plant proteomes in the tree were missing most or all mitochondrial sequences, and 10 were missing most or all chloroplast sequences (see **Table 3.2** and **Table S2**).

3.3.4 Homology inference and quantitative diagnostics

For each gene of the focal species, it is important to know which genomes in the clade tree do *not* have a homolog, as well as which ones do. `phylostratr` provides powerful diagnostic tree visualizations that, although they cannot infer the details of complex events, provide a strong starting point, and can reveal unusual cases of interest (**Figure 3.4**). The clade tree-heat map visualizations can indicate potential gene candidates for horizontal transfer, gene loss, or other confounding histories.

Horizontal gene transfer, in particular associated with endosymbiotic events, is observed widely across eukaryotes [40, 41, 42, 43], although it has been contested for humans [44]. Horizontally-transfered genes would be classified in the oldest strata to which they had a homolog by a standard BLAST-based phylostratigraphic analysis. For example, a gene present in a narrow eukaryotic clade but also with homology to a gene in some prokaryotes would be reported as a member of the phylostratum "cellular organisms". The clade tree diagnostics would reveal, through the absence of hits to intermediate strata, that this gene is a candidate for horizontal transfer from a prokaryote prior to the speciation event leading to the eukaroytic clade in which it is found.

In another example, the diagnostics for a gene that is correctly designated as ancient by `phylostratr` could reveal a further component to the story. A diagnostic pattern might show the gene to be present in some but not all clades. This indicates the possibility that the ancient gene confers a trait needed for survival only under particular conditions. It might be independently retained by clades that are exposed to those conditions, while lost in other clades where it is not needed.

Another approach to detect genes with irregular phylostratigraphic profiles is to find genes that “skip” strata. That is, genes that have no homologs in several intermediate strata, but do have homologs in more basal strata. An overview diagnostic summarizes these for each gene (*A. thaliana*) (**Table S4**) and *S. cerevisiae* (**Table S9**).

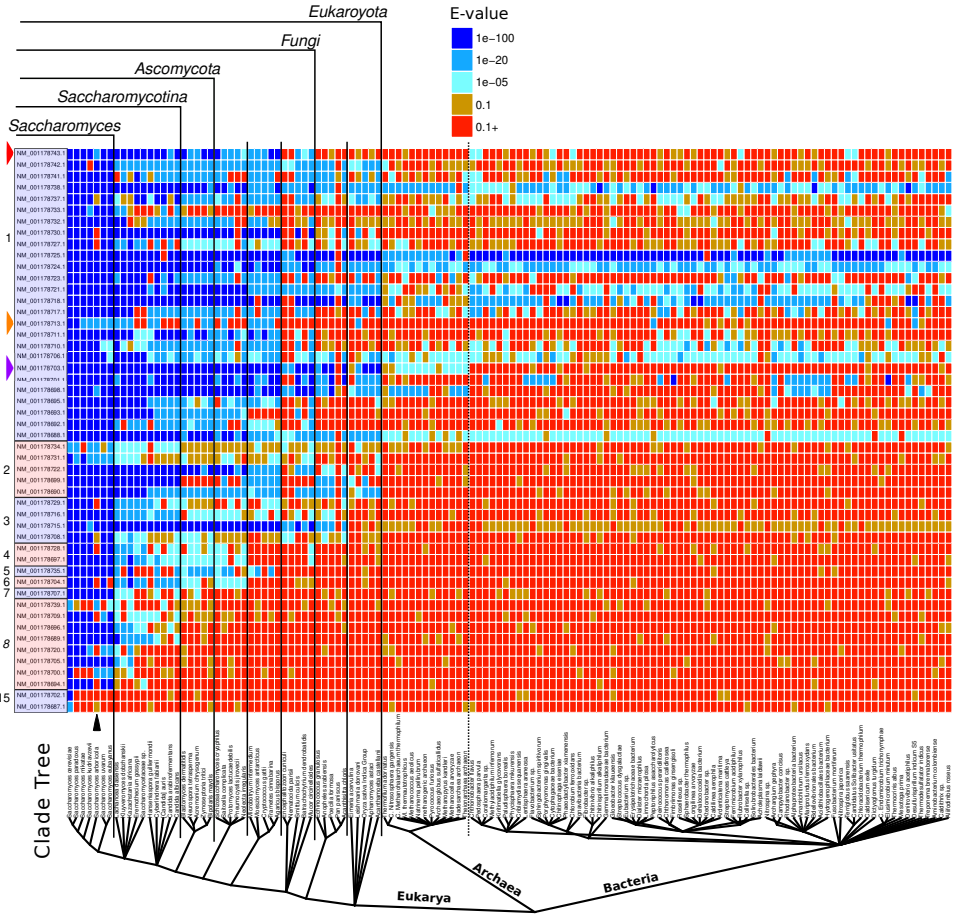


Figure 3.4 `phylostratr` gene by clade tree-heat map for *S. cerevisiae*. The figure (page 6 of a total of 234 pages) shows the degree of similarly of each protein encoded by the focal species (far left column) to that of its closest homolog in each species in the clade tree. Proteins encoded by consecutive *S. cerevisiae* genes (NM_001178687.1 - NM_001178743.1), sorted by phylostratum membership, are compared to the proteomes of 133 species. The designated phylostratum membership (1-15) is indicated on the far left. Red wedge: This gene has homologs in the fungal phylostrata, but not in other eukaryotes, and also has bacterial homologs; it is potentially associated with a horizontal transfer event. Orange wedge: A gene with skipped strata, possibly due to lateral transfer or false positive matches to prokaryotes. Magenta wedge: a gene present in eukaryotes and archaea but not in bacteria. Black arrowhead: The eight "missing" homologs in *S. arboricola* could be absent due to an interesting biological phenomenon such as a rapidly evolving genome or massive gene loss or due to an incomplete genome annotation; the gene annotation statistics for the *S. arboricola* genome assembly at NCBI support the latter explanation.

The diagnostic of **Figure 3.3D** globally visualizes the numbers of genes assigned to a given stratum and their next significant hit in an older stratum. About 80% of *A. thaliana* genes have inferred homologs in a given strata, and also in each of the more closely-related strata. This can be seen from the genes indicated in the boxes on the diagonal; genes represented on the diagonal are those where no phylostratum is skipped. About 20% of the genes in the *A. thaliana* study skip two or more sequential strata. Genes represented in cells off the diagonal indicate cases where a deeper stratum has an inferred homolog, but adjacent shallower branches do not.

Technical and/or biological factors could explain the phenomenon of "skipping". Skipped strata could result from poor quality data across all representatives of the skipped stratum. For example, 10 of the 13 mitochondrial genes annotated in *S. cerevisiae* skip four or more strata; this is likely due to missing mitochondrial genomes in many target species. Alternatively, skipped strata could appear due to false positives in the ancient stratum, possibly due to short repeats that inflate BLAST scores [45]. Such false positives likely explain, for example, the four genes that have significant hits within the Eukaryota and Brassicaceae strata, but no inferred homologs in intermediate strata (**Figure 3.3D**). There are also biological explanations for skipped strata, for example, multiple independent deletion events or a horizontal gene transfer [46]. Possible horizontal transfer events are reflected in the 1937 genes assigned to cellular organisms that skip Eukaryota, and whose next significant hits are in Viridiplantae (**Figure 3.3D**). Many of these genes are associated with photosynthesis and presumably resulted from the major endosymbiotic event by which a Viridiplantae ancestor acquired chloroplasts.

3.3.5 Future directions: Adding syntenic context to `phylostratr`

The flexible design of `phylostratr` provides options for additional functions to be added. Perhaps the most important of these would be to supplement the phylostratigraphic designation of a gene with its syntenic context. Syntenic context greatly reduces

the search space and thereby increases the sensitivity. This reduced search space permits finding sequence similarity at the DNA level, as well as at the protein level, thus potentially providing positive evidence of the *de novo* origin of a gene [12, 47, 43] versus the rapid evolution beyond recognition of an ancient gene [48]. The syntenic approach has its own limitations. 1) It is limited to closely related genomes. High resolution synteny maps are very tenable across recently-evolved clades in which synteny is maintained. However, because genome structure evolves dynamically [49], synteny maps between evolutionarily distant genomes are not feasible. 2) Synteny searches are limited to those regions in the genome that have synteny to another genome, thus genes or other features falling outside these regions cannot be evaluated.

Currently there is no general program for synteny-aware phylostratigraphy. A tool for syntenic analysis and its integration with `phylostratr` would provide a deeper understanding of the ontogenies of those genes that stem from the phylostrata across which synteny is preserved. This would provide understanding of the genomic origins of genes that originated at the accession/race, genus, or family level. It would also inform about the recent histories of older genes. Syntenic data could provide a systematic classification of genes according to their genomic context (e.g., near a transposon, near a recently duplicated gene, in a region of high GC content), which could be used to determine whether/how such features might be associated with evolution.

3.4 Conclusions

`phylostratr` is a highly automated and flexible platform for phylostratigraphic analysis that bypasses tedious data handling and provides an accessible, standardized framework. It provides a suite of tools to help researchers avoid common pitfalls in phylostratigraphy. `phylostratr` simplifies optimization of analysis parameters, and provides an unprecedented ability for rapid visual assessments. Ultimately, this will improve our

insight into gene ontogenies, and help uncover the richness and complexity of gene evolution.

`phylostratr` can be integrated naturally into other R-based bioinformatics and statistical pipelines. Conversely, because `phylostratr` is organized into core functions, individual modules can be repurposed for other applications. For example, the R algorithm used to select diverse species from a clade tree (Function 2 in **Figure 3.1**) can be used on its own and applied to diversify leaf selection in other types of trees.

The output of `phylostratr` may serve as the foundation for more specialized phylostratigraphic analyses. For example, age estimates from `phylostratr` can be passed to the `myTAI` R package [50] to explore potential events that correlate with new genes formation, and gain insight into whether some species show a large rise in new gene birth associated with a major environmental catastrophe or in relation to a large developmental shift like the evolution of seeds. Also, phylostratal assignments could be passed to the big-data JAVA-based MetaOmGraph (`https://github.com/urmi-21/MetaOmGraph`) to explore transcriptional patterns associated with genes assigned to a particular phylostrata.

3.5 Supplementary

The supplementary material for this chapter is available with the online publication. It is not included in this dissertation since it consists of many long tables that are needed only if the reader wishes to check my arithmetic.

Bibliography

[1] Jacob, F. (1977) Evolution and tinkering. *Science* 196, 1161–1166

[2] Kaessmann, H. (2010) Origins, evolution, and phenotypic impact of new genes. *Genome research* 20, 1313–1326

[3] Domazet-Lošo, T. *et al.* (2007) A phylostratigraphy approach to uncover the genomic history of major adaptations in metazoan lineages. *Trends in Genetics* 23, 533–539

[4] Carvunis, A.R. *et al.* (2012) Proto-genes and de novo gene birth. *Nature* 487, 370–374

[5] Tautz, D. and Domazet-Lošo, T. (2011) The evolutionary origin of orphan genes. *Nature Reviews Genetics* 12, 692–702

[6] Arendsee, Z.W. *et al.* (2014) Coming of age: orphan genes in plants. *Trends in plant science* 19, 698–708

[7] McLysaght, A. and Hurst, L.D. (2016) Open questions in the study of de novo genes: what, how and why. *Nature Reviews Genetics* 17, 567

[8] Šestak, M.S. and Domazet-Lošo, T. (2014) Phylostratigraphic profiles in zebrafish uncover chordate origins of the vertebrate brain. *Molecular biology and evolution* 32, 299–312

[9] Bhandary, P. *et al.* (2017) Raising orphans from a metadata morass: a researcher's guide to re-use of public 'omics data. *Plant Science*

[10] Khalturin, K. *et al.* (2009) More than just orphans: are taxonomically-restricted genes important in evolution? *Trends in Genetics* 25, 404–413

[11] Chen, L. *et al.* (1997) Evolution of antifreeze glycoprotein gene from a trypsinogen gene in Antarctic notothenioid fish. *Proceedings of the National Academy of Sciences* 94, 3811–3816

[12] Wu, B. and Knudson, A. (2018) Tracing the de novo origin of protein-coding genes in yeast. *mBio* 9, e01024–18

[13] Vakirlis, N.N. *et al.* (2017) A molecular portrait of de novo genes in yeasts. *Molecular biology and evolution* p. msx315

[14] Smith, S.A. and Pease, J.B. (2017) Heterogeneous molecular processes among the causes of how sequence similarity scores can fail to recapitulate phylogeny. *Briefings in bioinformatics* 18, 451–457

[15] Moyers, B.A. and Zhang, J. (2016) Evaluating phylostratigraphic evidence for widespread de novo gene birth in genome evolution. *Molecular biology and evolution* 33, 1245–1256

[16] Moyers, B.A. and Zhang, J. (2014) Phylostratigraphic bias creates spurious patterns of genome evolution. *Molecular biology and evolution* 32, 258–267

[17] Domazet-Lošo, T. *et al.* (2017) No evidence for phylostratigraphic bias impacting inferences on patterns of gene emergence and evolution. *Molecular biology and evolution* 34, 843–856

[18] Moyers, B.A. and Zhang, J. (2017) Further simulations and analyses demonstrate open problems of phylostratigraphy. *Genome Biology and Evolution*

[19] Casola, C. (2018) From de novo to "de nono": The majority of novel protein-coding genes identified with phylostratigraphy are old genes or recent duplicates. *Genome biology and evolution* 10, 2906–2918

[20] Jain, A. *et al.* (2018) The evolutionary traceability of proteins. *bioRxiv*

[21] Klasberg, S. *et al.* (2018) Origins and structural properties of novel and de novo protein domains during insect evolution. *The FEBS journal*

[22] Ekstrom, A. and Yin, Y. (2016) ORFanFinder: automated identification of taxonomically restricted orphan genes. *Bioinformatics* 32, 2053–2055

[23] Cheng, X. *et al.* (2015) A "developmental hourglass" in fungi. *Molecular biology and evolution* 32, 1556–1566

[24] Drost, H.G. *et al.* (2015) Evidence for active maintenance of phylotranscriptomic hourglass patterns in animal and plant embryogenesis. *Molecular biology and evolution* 32, 1221–1231

[25] Liebeskind, B.J. *et al.* (2016) Towards consensus gene ages. *Genome biology and evolution* 8, 1812–1823

[26] Pryszcz, L.P. *et al.* (2010) MetaPhOrs: Orthology and paralogy predictions from multiple phylogenetic evidence using a consistency-based confidence score. *Nucleic acids research* 39, e32–e32

[27] Consortium, U. *et al.* (2014) UniProt: a hub for protein information. *Nucleic acids research* p. gku989

[28] Federhen, S. (2011) The NCBI taxonomy database. *Nucleic Acids Research* 40, D136–D143

[29] Asara, J.M. *et al.* (2007) Protein sequences from mastodon and *Tyrannosaurus rex* revealed by mass spectrometry. *Science* 316, 280–285

[30] Neme, R. and Tautz, D. (2013) Phylogenetic patterns of emergence of new genes support a model of frequent de novo evolution. *BMC genomics* 14, 117

[31] Pearson, W.R. (2013) An introduction to sequence similarity ("homology") searching. *Current protocols in bioinformatics* 42, 3–1

[32] Krasileva, K.V. *et al.* (2013) Separating homeologs by phasing in the tetraploid wheat transcriptome. *Genome Biology* 14, R66

[33] Ewens, W.J. and Grant, G.R. (2006) *Statistical methods in bioinformatics: An introduction.* Springer

[34] Finn, R.D. *et al.* (2015) HMMER web server: 2015 update. *Nucleic acids research* 43, W30–W38

[35] Wickett, N.J. *et al.* (2014) Phylotranscriptomic analysis of the origin and early diversification of land plants. *Proceedings of the National Academy of Sciences* 111, E4859–E4868

[36] Matasci, N. *et al.* (2014) Data access for the 1,000 plants (1KP) project. *Gigascience* 3, 17

[37] Xie, Y. *et al.* (2014) SOAPdenovo-Trans: de novo transcriptome assembly with short RNA-seq reads. *Bioinformatics* 30, 1660–1666

[38] Johnson, M.T. *et al.* (2012) Evaluating methods for isolating total RNA and predicting the success of sequencing phylogenetically diverse plant transcriptomes. *PLoS One* 7, e50226

[39] Cheng, C.Y. *et al.* (2017) Araport11: a complete reannotation of the *Arabidopsis thaliana* reference genome. *The Plant Journal* 89, 789–804

[40] Méheust, R. *et al.* (2016) Protein networks identify novel symbiogenetic genes resulting from plastid endosymbiosis. *Proceedings of the National Academy of Sciences* 113, 3579–3584

[41] Yue, J. *et al.* (2012) Widespread impact of horizontal gene transfer on plant colonization of land. *Nature communications* 3, 1152

[42] Bock, R. (2017) Witnessing genome evolution: Experimental reconstruction of endosymbiotic and horizontal gene transfer. *Annual review of genetics* 51

[43] Thorpe, P. *et al.* (2018) Shared transcriptional control and disparate gain and loss of aphid parasitism genes. *Genome biology and evolution* 10, 2716–2733

[44] Salzberg, S.L. (2017) Horizontal gene transfer is not a hallmark of the human genome. *Genome biology* 18, 85

[45] Zhang, J. and Madden, T.L. (1997) PowerBLAST: A new network blast application for interactive or automated sequence analysis and annotation. *Genome Research* 7, 649–656

[46] Gao, C. *et al.* (2014) Horizontal gene transfer in plants. *Functional & integrative genomics* 14, 23–29

[47] Lu, T.C. *et al.* (2017) A comprehensive analysis of transcript-supported de novo genes in Saccharomyces sensu stricto yeasts. *Molecular biology and evolution* 34, 2823–2838

[48] Szczepaniak, A. *et al.* (2018) Legume cytosolic and plastid acetyl-coenzyme—a carboxylase genes differ by evolutionary patterns and selection pressure schemes acting before and after whole-genome duplications. *Genes* 9, 563

[49] Eichler, E.E. and Sankoff, D. (2003) Structural dynamics of eukaryotic chromosome evolution. *science* 301, 793–797

[50] Drost, H.G. *et al.* (2017) myTAI: Evolutionary transcriptomics with R. *Bioinformatics*

CHAPTER 4. `synder`: MAPPING INTERVALS BETWEEN GENOMES

Zebulun Arendsee, Andrew Wilkey, Urminder Singh, Li Jing, Manhoi Hur, Eve Syrkin Wurtele

Modified from a paper pre-released to BioRxiv.

Abstract

Ortholog inference is a key step in understanding the function of a gene or other genomic feature. Yet often no similar sequence can be identified, or the true ortholog is hidden among false positives. A solution is to consider the sequence's genomic context. We present the generic program, `synder`, for tracing features of interest between genomes based on a synteny map. This approach narrows genomic search-space independently of the sequence of the features of interest. We illustrate the utility of `synder` by finding orthologs for the *Arabidopsis thaliana* 13-member gene family of Nuclear Factor YC transcription factor across the Brassicaceae clade.

4.1 Introduction

A powerful first step in understanding the evolution and function of a genomic feature is resolving its genomic context, that is, comparing the feature to orthologous features in other species. Comparing multiple orthologous features across species allows evolutionary patterns to be uncovered. These patterns may include evidence of purifying selection, which implies the feature is important to the survival of the species; positive selection, implying the feature is rapidly evolving along one lineage; and functional dependencies between sites (for example, amino acids in an enzyme reaction site) [1]. These

evolutionary trends have direct application in fields such as rational protein design [2]. Distinguishing between orthologs (homologous features arising through speciation) and paralogs (homologous features arising through gene duplication) is foundational to understanding the history of a feature. Genomic context is also critical for discerning the origins of the often large numbers of species-specific "orphan" genes that are found in most genome projects [3, 4, 5, 6].

Identifying orthologs is not easy. A simple sequence similarity search of a query feature (e.g., a gene, transposon, miRNA, or any sequence interval) against a genome or proteome of a target species may obtain thousands of hits in a swooping continuum; these could include: the true ortholog, related family members (paralogs), and non-specific hits. Therefore, methods for winnowing the search results have been developed to identify the true orthologs. A straightforward approach to identify orthologs of protein-coding genes is reciprocal best hits [7]. In this technique, a protein encoded by a gene from the focal species is searched (e.g. with BLAST) against the target proteome. The highest scoring gene is then searched back against the proteome of the focal species. If the top scoring hit of the second search is the original query gene, then the two genes are accepted as orthologs. There are also methods that build on reciprocal best hits, such as the reciprocal smallest distance method that considers evolutionary distance in addition to similarity score [8].

Little or no significant sequence similarity is expected across species for some classes of features. A lack of significant similarity may stem from sequences being very short (e.g., a single promoter element or an miRNA) or it could result from very rapid mutation rates in the feature (e.g., intragenic intervals that are under little or no purifying selection). Orphan genes, which by definition have no protein homolog in related species, are an example of a feature for which sequence comparisons alone cannot delineate the region in the target genome from which the orphan gene arose [4]. These genes are often both short *and* rapidly evolving, making it very difficult to find orthologous genomic

regions (possibly non-coding) even in closely-related target species. Without an ability to identify orthologous genomic intervals, the pathway of evolution of an orphan cannot be determined; for example, orphans of *de novo* origin cannot be distinguished from those orphans that stem from rapid mutation [3].

Purely sequence-based methods are also problematic if the true ortholog of a query gene is duplicated in the target species. In this case, the target species contains two genes that are true orthologs of the query gene. The co-evolution of duplicate genes relative to their singleton ortholog, is of interest in theoretical evolution [9]. One of the copies may rapidly evolve to gain a new function or it may become a pseudogene [10]. In either case, the reciprocal best hits method would find only the conserved copy.

A different approach to ortholog identification, one which does not depend on the genome-wide sequence similarity of the query features themselves and that does handles duplication events, is to consider the genomic context of the query, i.e., synteny [11]. Genomic synteny is the conservation of the order of genomic features between two genomes [12].

The most obvious approach to a context-based search is to include the flanking regions of a query feature of interest when searching for an ortholog in the target genome. This approach is used by MicroSyn [13] for finding orthologs of features, such as miRNAs, that are too short and numerous to be easily searched by their sequence alone. While this approach works well for an individual query feature, extending it to a high-throughput analysis is problematic, since no single cutoff for flank length will work well for all cases. For instance, a sequence residing within a highly repetitive centromere might require flanks of megabases.

An alternative to looking at the flanking sequence of each query feature individually is to reference a genome-wide synteny map. Rather than searching for the feature directly, orthologs of flanking syntenic regions (blocks) can be identified, and a potential ortholog of the query feature can be identified in the target genome by searching within

the syntenic region. This strategy has been applied to study the genomic origin of orphan genes [14] (and the method refined in [15]) where a map of one-to-one orthologous genes was used to infer the orthologous genomic intervals where the non-genic sequence corresponding to the orphan genes is expected to reside. The one-to-one map made the computational problem very easy, and could effectively identify a sub-set of the genes of *de novo* origin, but the map was very coarse, especially in regions of low gene density, so no information is obtained for other orphan genes.

There are many programs designed to build synteny maps. Some programs build sparse synteny maps from given sets of orthologous genes (e.g., OrthoClusterDB [16]). Others perform full genome alignments. Of these, some focus on large scale (megabase range) syntenic blocks that are conserved across great evolutionary distances, while others focus on micro-synteny, producing maps of many small syntenic blocks that capture local inversions, duplications and deletions (e.g., `BLASTZ` [17], `MUMmer4` [18], and `Satsuma` [19]). These micro-synteny programs are of greatest interest in this paper.

The diverse synteny mapping tools, though highly variable in granularity and accuracy [11, 20], provide powerful approaches to enable the study of comparative genome evolution [21, 22, 12, 23] and to glean novel information about the origin of *de novo* orphan genes [24, 25, 14, 26, 15]. However, the use of these maps as a tool for orthology has been generally limited to either manual inspection, or to considering only those query features that overlap syntenic blocks.

`synder` is designed to infer orthologous regions in the target genome, even when the orthologs are *between* syntenic blocks, and to assess the quality of the inferences. To do this, it traces query features from a focal genome to a target genome using a whole-genome map. `synder` is a high-performance program with a core written in C++ and an R wrapper for integration into R workflows. It will work with any synteny map, but was designed for fine-grain micro-synteny maps that capture local inversions and transpositions. It assembles collinear sets of syntenic blocks from the map and uses them

to infer tight search intervals for each query on the target genome, naturally handling duplication events and inversions. `synder` also provides detailed information about the quality of the search result. The only input required is a whole-genome synteny map and a set of features of interest in the focal genome. Thereby, `synder` automates the use of syntenic information to study orthologs across any pair of species with sufficiently conserved synteny.

4.2 Algorithm

Table 4.1 Terminology

focal genome:	The genome that contains the query features.
target genome:	The genome in which search intervals are found for each feature of interest.
query feature:	The sequence interval delineating any feature of interest in the focal genome. This could be a protein-coding gene, an miRNA, an intron, a transposon, a nucleotide repeat, an lncRNA or any other genomic feature
blocks:	Focal and target genome intervals that are inferred to be orthologs by an outside synteny program.
synteny map:	A set of blocks for a pair of genomes.
syntenic interval:	A single interval on one side of a synteny map.
adjacent intervals:	Two syntenic intervals on the same scaffold with no syntenic interval located entirely between them.
collinear blocks:	Two blocks where both the focal and target syntenic intervals are adjacent and in the same orientation.
collinear block set:	An ordered set of blocks where block i is collinear to block $i+1$.
query context:	All blocks that overlap or are adjacent to the query interval.
search interval:	An expected location of an ortholog of a query feature in the target genome.
search space:	The union of search intervals for a given query interval.
synteny score:	A score for a syntenic block produced by the outside synteny program.
synder score:	A score for the relative reliability of a search interval (see **Figure 4.5**).

The primary function of `synder` is to map a user-designated set of query features in the focal genome to a set of search intervals in a target genome (see **Table 4.1** for

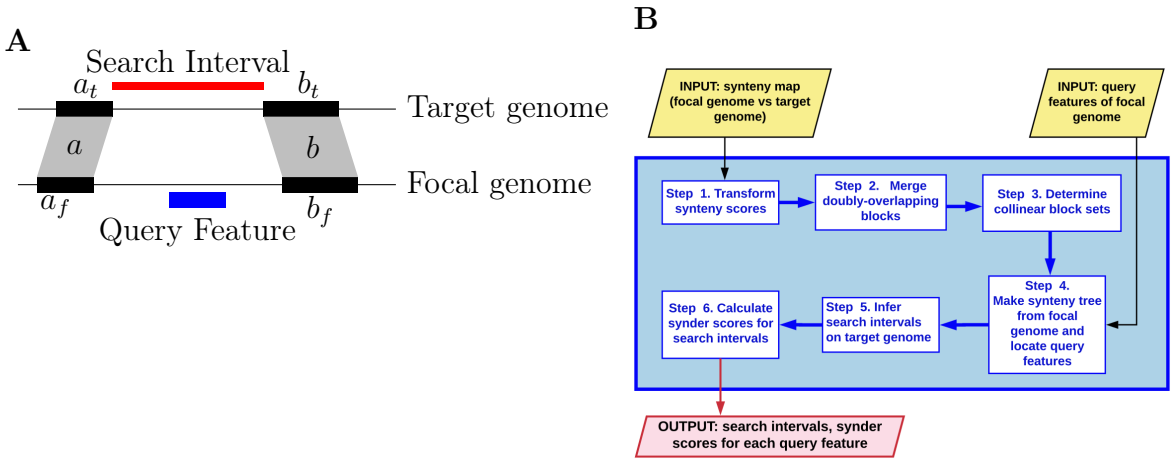


Figure 4.1 The `synder` algorithm identifies search intervals for query features based on synteny. **(A)** Diagram of a very simple syntenic relationship across focal and target genomes. a_f, a_t, b_f, and b_t are four syntenic intervals that comprise block a (a_f, a_t) and b (b_f, b_t). Blocks a and b are collinear and provide landmarks for associating the **query feature** in the focal genome with its **search interval** in the target genome. **(B)** Flow chart of the steps in the `synder` algorithm. `synder`: 1) transforms the synteny scores for each of the blocks in an input synteny map, such that scores are additive; 2) merges doubly-overlapping blocks; 3) assigns each block in the synteny map to exactly one collinear set of blocks; 4) finds the overlapping or nearest flanking syntenic intervals for each query feature in the focal genome (e.g., a_f and b_f in **A**); 5) for each query feature (i.e. the interval corresponding to a feature of interest on the focal genome) finds all collinear block sets that contain at least one of the blocks that flank or overlap the query feature, and then relative to each of these collinear block sets, maps the query interval to a search interval in the target genome; and 6) calculates search interval scores for each query feature relative to the search interval of each collinear set. The final output provides the query features with their corresponding hits in the target genome and a composite score for each hit.

terminology and **Figure 4.1** for overview). To do this, `synder` contextualizes the query features based on a user-provided synteny map for the focal and target genomes. Query features of the focal genome are mapped to an associated synteny-based search interval on the target genome; this search interval delineates the region of the target genome where the query feature is predicted to be located.

Algorithm 1 is an overview of the `synder`'s search algorithm. Each of the functions in this algorithm is defined in detail in the subsequent sections.

```
1  function synderSearch(q⃗, M, r, k, t):
2      s⃗ = transformScores(M, t)
3      M′ = mergeOverlapping(M, s⃗)
4      C = collinearSets(M′, k)
5      T = buildTree(M′)
6      foreach q in q⃗ do
7          a⃗ = anchors(q, T)
8          c⃗ = searchSets(a⃗, C)
9          foreach c in c⃗ do
10             (i, b) = searchInterval(q, c)
11             s = score(q, c, r)
12             recordRow(q, i, s, b)
13         end
14     end
```

Algorithm 1: A high-level overview of the core `synder` search algorithm. $\vec{q}$, list of query features; M, synteny map; r, search interval score decay rate (see **Figure 4.5**); k, number of interrupting blocks that is tolerated; t, type of synteny score; $\vec{s}$, vector of transformed, additive scores used in assigning final scores to each search interval. `synder` transforms scores and merges overlapping blocks to yield a processed, reduced synteny map, M'. Sequential syntenic blocks, C, are determined from M'. T, the interval tree data structure, is then used to find the syntenic context (i.e., the anchors, $\vec{a}$) on the focal genome for each query feature, q. Next, query features are mapped to one or more collinear set of blocks, $\vec{c}$. For each block, the associated search interval, i, is identified and the type of boundary, b, is determined. Each search interval is given a `synder` score, s. Finally, each search interval is recorded in the output table as a single row including the query feature (q), search interval (i), synder score (s), and search interval type (b).

4.2.1 Input Synteny Map

The primary raw input to `synder` is a synteny map that is provided as a table where each row describes one block. Each block consists of: an interval in the focal genome; an inferred syntenic interval in the target genome; a synteny score representing some metric of the confidence that the pair of intervals is orthologous; and, the relative orientation of the intervals. The focal and target intervals are each described by a chromosome/scaffold name and a start and stop position. The synteny score for each block is some measure of quality/certainty (e.g., percent identity or p-value) that is specific to the tool that used to generate the map. The orientation of the block is the strand in the target genome relative to the query, with '+' indicating the same strand and '-' indicating the inversion.

4.2.2 Step 1. Transform synteny scores

The synteny scores for the blocks in a synteny map may be expressed in a variety of ways by the various synteny programs. Strong similarity may be represented by low numbers (e.g., if scores are e-values) or high numbers (e.g., if scores are bitscores). Scores may be additive (e.g., bitscores) or averaged (e.g., percent similarity). The user must specify the type of the input synteny scores. Internally, the `synder` algorithm transforms these scores so that they are additive. More specifically, `synder` assumes $S(a+b) = S(a) + S(b)$, that is, if the blocks a and b are concatenated, then the synteny score should be equal to the sums of the scores for blocks a and b. `synder` transforms the synteny map scores to an additive score using one of the transforms below:

$$\text{transformScore}(s, l) = \begin{cases} \text{score density} & l * s \\ \text{percent identity} & l * s/100 \\ \text{e-value or p-value} & -log(s) \\ \text{otherwise} & s \end{cases} \tag{4.1}$$

Where s is the input synteny score and l is the interval length. `synder` transforms the scores when it loads a synteny file, updates them as needed in Step 2, and ultimately uses them in Step 6 to generate scores for the final search intervals.

4.2.3 Step 2. Merge doubly-overlapping blocks

In a "perfect" synteny map, blocks would not overlap on both the focal and target sides. In practice, however, synteny algorithms occasionally produce overlapping blocks. These cases would produce multiple collinear block sets that have the same orientation and cover the same region. To avoid this, `synder` merges any blocks that overlap on both the focal and target sides. The interval of the merged blocks is the union of the overlapping block intervals. The `synder` score of the merged blocks is calculated by summing the non-overlapping interval scores with the maximum of the overlapping intervals:

$$S_{ab} = d_a(l_{a_f} - l_o) + d_b(l_{b_f} - l_o) + l_o \max(d_a, d_b) \tag{4.2}$$

Where d_a and d_b are the score densities of blocks a and b (density is the synteny score for a block divided by the length of the syntenic interval on the focal genome); and where l_{a_f}, l_{b_f} and l_o are the lengths of a, b, and their overlap, respectively.

A potential downside of this approach is that, when more than two intervals are doubly-overlapping, the order in which the scores are merged matters, with blocks merged later having a stronger influence. A second issue is that the merged score is calculated based on the intervals on just one side of the synteny map. The length of each interval, and the length of the overlap between the intervals, may vary between the two sides of the synteny map. For now, we do not address either of these issues, since doing so would complicate the algorithm and probably have little effect on any biological dataset (since doubly-overlapping intervals are uncommon).

4.2.4 Step 3. Determine collinear block sets

`synder` assigns each block in a synteny map to exactly one set of collinear syntenic blocks (**Figure 4.2**). Each collinear block set consists of adjacent blocks that are ordered on the query and target sides. `synder` considers two syntenic intervals "adjacent" if they are on the same scaffold and no syntenic interval is contained entirely between them. Adjacency on the target-side further requires that the intervals have the same orientation (+/-) relative to the focal genome. Two blocks are collinear if the syntenic intervals on the focal-genome and target-genome are adjacent. The collinear block sets may be inverted and/or may overlap other collinear block sets on either the focal or target side (e.g., for duplicated sequences). The individual blocks that make up the collinear set are used in Steps 4 and 5 to delimit the search intervals on the target genome relative to each query feature of the focal genome.

This approach can be overly strict, resulting in many small collinear block sets. Tracing blocks across whole genome duplication, and subsequent genome alterations [27], is particularly challenging, since intervals in the homologous chromosomes could randomly diverge, resulting in a synteny map that alternates between mapping to one chromosome and the homologous chromosome. This is especially problematic in plants, where whole genome duplications are common [28]. To reduce this potential complication, `synder` provides the user an option to relax the adjacency restriction by allowing k syntenic intervals that map to alternative target scaffolds to interrupt a pair of query-side intervals in a collinear set.

The output of Step 3 is the set of blocks that are non-overlapping and adjacent. Each block is assigned to exactly one collinear block set.

4.2.5 Step 4: Find focal genome contextual anchors for query features

The next step is to find the blocks that contain, overlap, or are adjacent to each query feature on the focal genome. These blocks will provide "anchors" that will be used

in Step 5 to map the query feature to one or more collinear sets of blocks in the target genome and hence to identify the search interval(s) in the target species.

The user provides the query features as a Gene Feature Format (GFF) file that describes the genomic intervals on the focal genome corresponding to the query features of interest. A modification is to provide the GFF file along with BLAST or other whole-genome similarity scores; this modification was used as input for the case study on the NF-YC gene family (see RESULTS section).

```
1  function buildTree(q⃗):
2      if length(q⃗) == 0 then
3          return Null
4      end
5      c = midpoint(q⃗)
6      v⃗_left = filter(q⃗, λ q → c < q_1)
7      v⃗_mid = filter(q⃗, λ q → q_1 ≤ c ≤ q_2)
8      v⃗_right = filter(q⃗, λ q → q_1 < c)
9      T_left = buildTree(v⃗_left)
10     T_right = buildTree(v⃗_right)
11     return Tree(c, v⃗_mid, T_left, T_right)
```

Algorithm 2: Build a synteny interval tree. `buildTree` takes a vector of intervals, $\vec{q}$, on a given scaffold/chromosome of the focal genome and returns an interval tree data structure. The midpoint c is an integer equal to the middle position in the interval in the middle of the vector of intervals (by index). If the input vector is sorted, then the midpoint will tend to be near the center of the scaffold. `filter`$(\vec{q}, f)$ selects the subset of intervals in $\vec{q}$ for which the condition $f(q)$ is true. The filters in lines 6-8 partition each element in $\vec{q}$ into one of three sets: intervals on the left of the midpoint c, intervals overlapping the midpoint c, and intervals on the right of the midpoint c. New trees are created recursively for the left (less than) and right (greater than) sets of intervals. `buildTree` returns a new syntenic interval Tree object, (T), that stores the midpoint c, all overlapping syntenic intervals $(\vec{v}_{mid})$, and the left and right child trees. The Tree will be used in **Algorithm 3** to identify the anchors for each query feature.

`synder` uses a modified interval tree algorithm to locate the syntenic intervals on the focal genome that "anchor" the query feature. Building the interval tree is an $O(n \log(n))$ operation (see **Algorithm 2**) and searching for a given interval is $O(\log(n) + m)$, where m is the number of overlapping intervals returned and n is the size of the synteny map

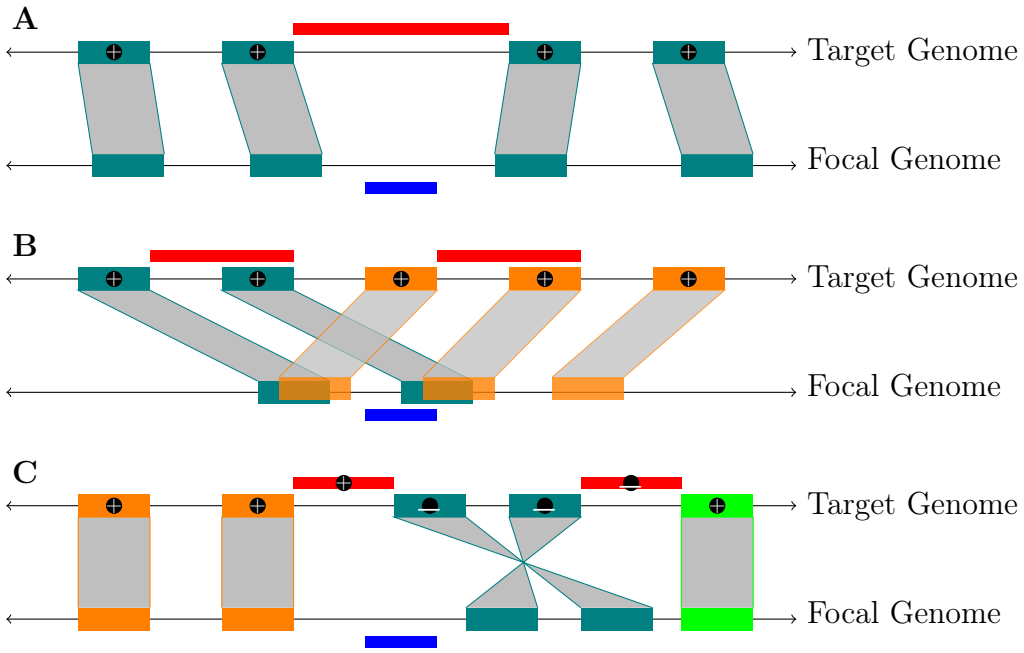


Figure 4.2 Collinear block set construction with focal-genome "anchors" to infer search intervals on the target genome. The **blue bars** below each focal genome are the query features. Genome-wide collinear block sets (colored orange, teal or green) are identified in Step 3, and are used to identify the focal-side anchors for each query feature in Step 4. The **red bars** above the target genome are the search intervals inferred by `synder` in Step 5. **(A)** a simple case where the query feature does not overlap a syntenic interval and is bound between syntenic intervals in a collinear set of blocks. **(B)** a tandem duplication where `synder` resolves the blocks into two collinear block sets (teal and orange) and infers search intervals for each. **(C)** a query feature that is **unbound** on each side (see **Figure 4.4**) resulting in one search interval relative to the orange collinear block set (red bar on left) and one search interval relative to the inverted teal block collinear set (red bar on right). (+/-) signs in the target search intervals represent their strand orientation relative to the query ('-' is an inversion).

(see **Algorithm 3**). We modified an algorithm that returns only directly overlapping intervals [29], to enable `synder` to find the flanking intervals (upstream and downstream intervals) when no overlapping intervals are found (see **Figure 4.3**).

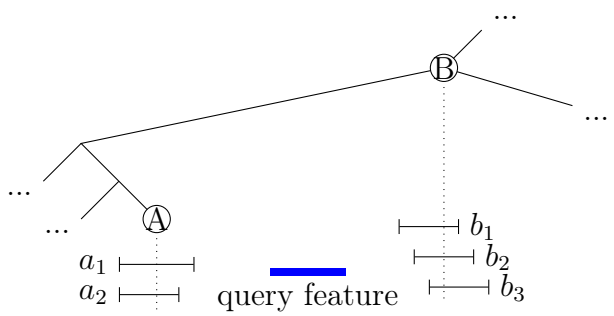


Figure 4.3 Identification of the syntenic intervals that anchor a query feature on the interval tree from the focal genome. A and B are nodes in the interval tree. A stores the overlapping syntenic intervals a_1 and a_2. B stores the overlapping intervals b_1, b_2 and b_3. The query feature falls between the syntenic intervals stored in nodes A and B. The interval tree algorithm first makes the tree, and then finds the nearest node to the query. If the closest node found is A, and the nearest syntenic interval (a_1) detected is on the left side of the query feature, then the algorithm will trace the tree until it finds the first node to the right of the query feature (i.e., node B). Conversely, if the closest node found is to the right of the query feature (node B), then the tree is traced one branch to the left, and then as many branches to the right as possible (i.e., until node A is found). In either case, all overlapping nodes in A and B are returned as the anchors for this query sequence.

4.2.6 Step 5. Map query features and infer target genome search intervals

Each query feature is mapped to a search interval that is created with respect to each associated collinear block set (**Figure 4.2**). To do this, `synder` first classifies each edge of the query feature relative to its relationship to an associated collinear set of blocks (**Figure 4.4**), where each query feature edge may fall: 1) between two collinear sets of blocks (**unbound**); 2) inside a block (**inblock**); 3) between blocks comprising a collinear block set (**bound**); or 4) beyond all blocks, i.e., near the beginning or end of the scaffold (**extreme**). `synder` sets each boundary of the target genome search interval to the nearest edge of a block in the collinear set if the edge is **inblock** or **bound**; to

```
1  function search(T, t, q):
2      if t = Null then
3          return Empty
4      end
5      r⃗ = []
6      if t_midpoint < q_1 then
7          foreach i in stopSorted(t) do
8              if q_1 ≤ i_2 then
9                  r⃗.add(i)
10             end
11         end
12         if t_right = Null and length(r) = 0 then
13             r⃗.add(opposite(T, t))
14         end
15         r⃗.add(search(T, t_right, q))
16     end
17     else if t_midpoint > q_2 then
18         foreach i in startSorted(t) do
19             if q_2 ≥ i_1 then
20                 r⃗.add(i)
21             end
22         end
23         if t_left = Null and length(r) = 0 then
24             r⃗.add(opposite(T, t))
25         end
26         r⃗.add(search(T, t_left, q))
27     end
28     else
29         foreach i in startSorted(t) do
30             r⃗.add(i)
31         end
32         r⃗.add(search(T, t_left, q))
33         r⃗.add(search(T, t_right, q))
34     end
35     return r⃗
```

Algorithm 3: Given the input of a syntenic interval tree (T), the current node in the tree (t), and a query feature (q), find all syntenic intervals in the focal genome that overlap the query feature. $\vec{r}$.`add(`$\vec{x}$`)` means intervals $\vec{x}$ are added to the search result $\vec{r}$. q_1 and q_2 represent the left- and right-hand edges of the query feature. i is an interval in the interval tree. $t_{midpoint}$ is the midpoint of the current node in the tree. t_{left} and t_{right} and the left- and right-hand subtrees. If no intervals are found, the `opposite` function returns the nearest blocks on each side of q (see **Figure 4.3**).

the nearest syntenic interval beyond the collinear set if the edge is **unbound**; or to the end of the scaffold if the edge is **extreme** (see **Figure 4.4**).

If a search interval is bound by two blocks, a and b, which define the two bounding intervals, $[a_1, a_2]$ and $[b_1, b_2]$, then the search interval be the inclusive interval $[a_2, b_1]$. We use an inclusive interval, rather than the exclusive interval from $(a_2 + 1)$ to $(b_1 - 1)$, to avoid negative length intervals that would occur when $b_1 = a_1 + 1$ (as would occur if there is a deletion in the target genome).

function `score`(q, $\vec{b}$, r)**:**
  s = 0
  foreach b *in* $\vec{b}$ **do**
    o = `overlap`(q, b)
    $d = b_{score}/(b_2 - b_1 + 1)$
    s += d * $(o_2 - o_1 + 1)$
    if $b_1 < q_1$ **then**
      s += $d \int_{q_1 - b_1}^{\max(0, q_1 - b_2)} e^{rx} dx$
    end
    if $b_2 > q_2$ **then**
      s += $d \int_{b_2 - q_2}^{\max(0, b_1 - q_2)} e^{rx} dx$
    end
  end
  return s

Algorithm 4: Calculating the *synder score* (s) for a query feature and the set of collinear blocks from the target genome to which it is anchored. q is the query feature, b is a focal genome-side syntenic interval within collinear block set $\vec{b}$, and o is the intersection (of zero length or greater) between q and b. The start and end points (edges) of the query feature q are q_1 and q_2 (as for edges of b and o). b_{score} is the synteny score associated with syntenic interval b. r is an adjustable parameter, the decay rate.

4.2.7 Step 6. Score collinear sets relative to overlapping query features

`synder` calculates the *synder score* for each input query feature relative to each associated collinear block set (**Figure 4.5**). The score reflects the intuitive ideas that: 1) query features are more reliable if a greater proportion of their sequence overlaps blocks in a collinear set; and, 2) query features are more reliable if they are within collinear

block sets that are densely packed. In cases where many possible search intervals are identified for a given query feature, the *synder scores* can be used to compare the relative quality of the search intervals. The `synder` score is especially important when k (the number of interrupting blocks that is tolerated) is high, since a large k allows large gaps between blocks in the collinear block sets. The pseudocode for `synder`'s scoring algorithm is shown in **Algorithm 4**. Note that input scores for syntenic blocks have been transformed to be additive (**Equation 4.2.2** (Step 1)).

In **Algorithm 4**, each block in the collinear block set can contribute to the total *synder* score (**Figure 4.5**). The score decay rate is controlled by the adjustable parameter r. For the default settings, the weight of the scores of blocks that neither overlap nor partially overlap the query feature decays exponentially with the absolute distance from the nearest query feature bound on the focal side. If the user sets r to be a low positive number, the weight at a given position will fall slowly with distance from the query interval (e.g., when $r = 0.001$ the weight will fall by half by 1000 bases from the nearest query feature bound); thus, all blocks in the collinear set will contribute to the score, but they matter less with distance (**Figure 4.5**, $r > 0$). $r = 0$ would give equal weight to all blocks in the collinear set, in that case, the density of the map will not affect the score, and the score would simply be equal to the sums of the total scores for all the syntenic blocks. A high value, such as $r = 100$, would completely ignore genomic context, basing the query feature score only on the portions of syntenic blocks that overlap the query feature. With this r setting, the `synder` score would be 0 if the query feature does not overlap any syntenic block.

4.3 Results and discussion

Mapping genes in a gene family in one species to their orthologs in a related species is a major usage case for `synder`. To demonstrate the use of `synder`, we identify orthologs

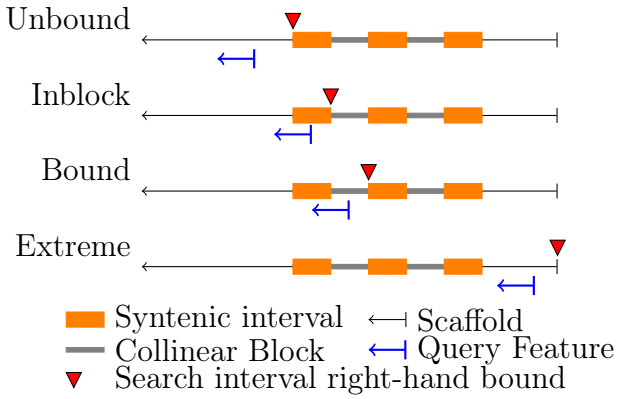


Figure 4.4 Snapping rules to define the location of the search interval edges on the target genome. 1) The left and right edges of the query feature are used to define the search interval relative to a given collinear block. Only the target genomes and the right edge of the query featured (represented by perpendicular line on query feature) are shown. 2) The right-hand edge of the search interval is then assigned (red triangles) (Rules are the same for the left edge). **Unbound**: the edge does not overlap the collinear block set. **Inblock**: the edge is inside a syntenic interval. **Bound**: the edge is between intervals in a collinear block set. **Extreme**: the edge is beyond any syntenic interval (near end of scaffold).

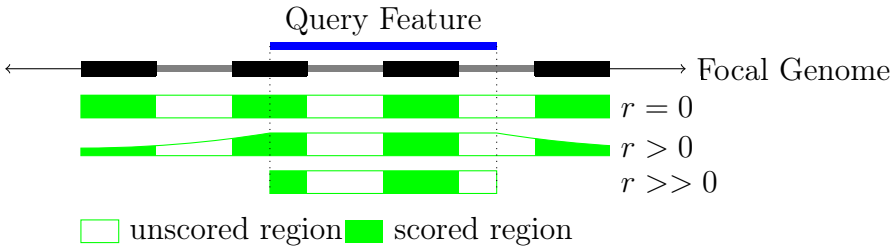


Figure 4.5 Calculation of synder score for a query feature relative to a collinear block set. **Black bars:** the collinear set of syntenic blocks that anchor the focal genome to the target genomes. (Only depicted on focal genome.) The total score of the search interval is the sum of the scores for each block. The score for each block, relative to the query feature (blue bar), is equal to the synteny score for the block times the "weight" of the block (determine by the adjustable parameter, r). Three values for r are depicted. The weight of each block is the area represented by the **solid green**. The intervals between blocks (**empty green**) do not contribute to the score.

of the *A. thaliana* NF-YC family genes across several species in Brassicaceae (**Figure 4.6**).

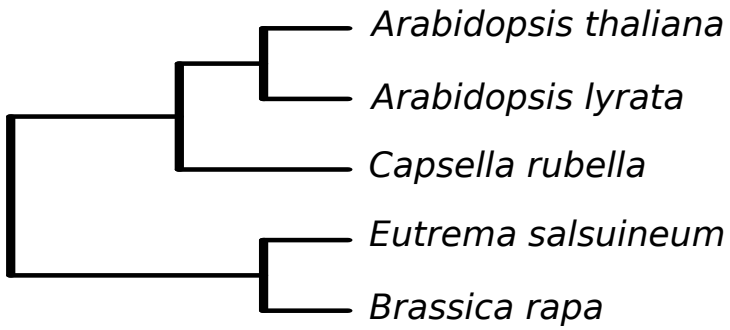


Figure 4.6 The species tree of the Brassicaceae used in this study. *A. thaliana* is the focal species.

.

The canonical NF-Y is a heterotrimeric, eukaryotic transcription factor that is comprised of subunits NF-YA, NF-YB and NF-YC [30]. NF-Y has been associated with characteristics as diverse as cell division [31], cancer [32, 33], drought tolerance [34], broad spectrum disease resistance [35], and carbon and nitrogen partitioning [36]. In animals and fungi, a single gene encodes each of the three NF-Y subunits. In contrast, the NF-Y subunits in plants are each encoded by gene families of 10-15 genes [33, 37, 38]. The combinatorial complexity of the potential plant NF-Y complexes that could be formed from the three NF-Y subunits has obfuscated the role of each individual family member, although progress is being made. For example, NF-YA1 of alfalfa controls successful symbiosis between rhizobia and plant, and is required for the persistence of the nodule meristem [39, 40].

Compounding the complexity of the action of NF-Y subunits in the cannonical heterotrimer, specific family members of at least two of the subunits, NF-YC and NF-YB, can also form associations with a variety of other nuclear proteins [41, 42, 36]. One of three NF-YC proteins can interact with one of two NF-YB proteins to enable CONSTANS-promoted photoperiod-induced flowering [43, 44]. NF-YC4 of *A. thaliana* interacts with the protein of the orphan gene QQS to modulate carbon and nitrogen

partitioning [36]. Several NF-YC family members interact with histone deacetylase 15 (HDA15) in the light to reduce histone acetylation, which in turn decreases hypocotyl elongation [41].

Clear determination of NF-YC orthologs across species would permit the assessment of the relationships among orthology and function in evolution of this gene family. In practice, ortholog identification is often based only on sequence similarity scores. The highest scoring match, however, may not be the true ortholog. `synder` allows more reliable ortholog inference by finding the similarity matches that overlap the inferred syntenic regions. In this way, `synder` may serve as a syntenic filter downstream of the similarity search.

4.3.1 NF-YC orthologs: *Arabidopsis thaliana* compared to *Arabidopsis lyrata*

The specific case of determining the *A. thaliana* NF-YC orthologs in its sister species, *A. lyrata*, illustrates the use of `synder` in resolving orthologs. Since NF-Y is a large family, the paralogous NF-YC family members must be distinguished from the true orthologs.

The genomic relationship between the two species further challenges analysis. While the species diverged only about 8.8 million years ago [45], *A. lyrata* has undergone a whole genome duplication since splitting from the common ancestor it shares with *A. thaliana*. This complicates orthology inference, since each *A. thaliana* gene is expected to have two orthologs in *A. lyrata*. Only one of each duplicate *A. lyrata* ortholog may have preserved a function. Its sister ortholog may have been deleted or become a pseudogene through genome fractionation [27, 46]. Alternatively, a sister ortholog may have undergone selection for a completely new molecular function [47, 48]. A third possibility is that the molecular function of each sister ortholog was preserved through the neutral process of subfunctionalization [47].

In this analysis, we built a synteny map with Satsuma [19] using default parameters that yielded 229,562 syntenic blocks with a median length of 163 nt (1st quantile = 33 bases, 3rd quantile = 365 bases). This is a very dense map: the *A. thaliana* genome is about 120M in length, thus there are an average of around 1900 syntenic blocks per megabase.

A `synder` search mapped 12 of the 13 query NF-YC family member genes to a search interval in *A. lyrata* that also contained the top BLAST hit (see Hits worksheet in supplementary) (**Figure 4.7**). Further, `synder` uniquely mapped 11 of the 13 query genes to a single *A. lyrata* gene. NF-YC5 and NF-YC10 are mapped by synder to two genes in *A. lyrata*, potentially reflecting that the genome duplication of *A. lyrata* was syntenically conserved. In contrast, a BLAST search yielded a nearly fully connected graph between NF-YC members in the two species. In the case of NF-YC8, `synder` identified orthologs that were located in the same syntenic region of the two genomes; however,the whole genome BLAST did not identify these likely orthologs, but rather other sequences were among the top hits. `synder` and BLAST identified the same gene as having the highest score. A second NF-YC12 ortholog was identified by `synder` that was not selected by BLAST.

If each pair of orthologs in *A. lyrata* had undergone only minimal sequence divergence and if synteny was maintained in each case, a `synder` analysis might uniquely identify two *A. lyrata* orthologs for each of the 13 NF-YCs of *A. thaliana*. Indeed, `synder` identified two hits for three of the family member: NF-YC5, NF-YC10 and NF-YC12. The BLAST results cannot reveal whether the second top BLAST hit is an ortholog or not.

4.3.2 NF-YC orthologs across the Brassicaceae family

This approach can be easily extended across the Brassicaceae family. We consider the species in **Figure 4.6**. In each species, tBLASTn alone links each NF-YC gene to nearly all of the other NF-YC genes; in contrast, `synder` identifies unique mappings to orthologs,

many of which differ from the highest BLAST hit. As syntenic distance increases, the orthologs become more difficult to identify through syntenic methods (**Figure 4.7**). However, for those genes located in syntenically-conserved regions, `synder` would more reliably identify the ortholog. For example, in *B. rapa*, `synder` identifies an NF-YC ortholog within the syntenic search space for NF-YC3, NF-YC5, NF-YC7, and NF-YC8 that does not correspond to the top BLAST hit (**Figure 4.7**). `synder` identified a syntenic ortholog of NF-YC6 that is only a weak BLAST hit. For NC-Y2, the ortholog identified by `synder` is not annotated at all in *B. rapa*. It may or may not be an expressed gene, but it is most likely an ortholog. Thus, `synder` can be used to augment whole-genome similarity inferences with syntenic context information.

4.4 Conclusion

`synder` provides a flexible, reproducible method to track specific genetic events. It also provides a pathway to evaluate broad biological concepts, including the evolution and diversification of gene families; the predominant mechanisms of diversification across lineages of eukaryotes and prokaryotes; the effects of genome duplication; and the relationship of different features to genomic instability. These types of analyses can ultimately reveal those genetic events that might be associated with particular evolutionary consequences, such as rapid evolution, horizontal transfer, *de novo* emergence of genes, transposition, or duplication.

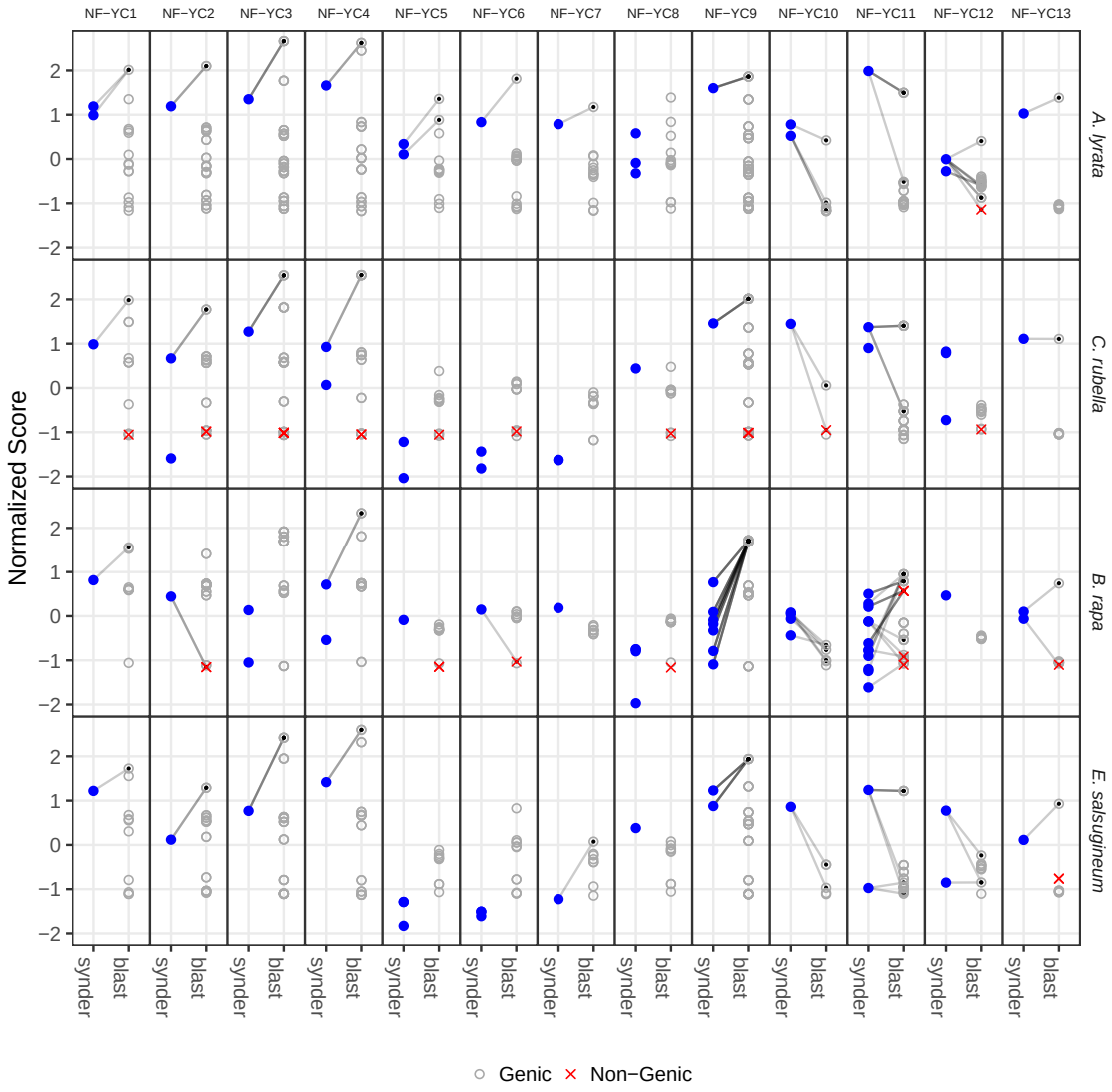


Figure 4.7 Comparison of orthologs inferred by `synder` and BLAST for the 13 *A. thaliana* NF-YC family members across genomes of four target species from Brassicaceae. Each row represents predicted orthologs in a target species. The x-axis for each box compares `synder` and tBLASTn scores. **Blue dots**, search intervals on the target genome that overlap a gene; **gray circles**, tBLASTn hits (E-value<0.001) on the target genome; **red X's**, tBLASTn hits on the target genome that do *not* overlap any annotated gene. The Normalized Score (y-axis) is the score for `synder` search intervals and tBLASTn hits; synder scores were logged, and tBLASTn E-values were transformed with a negated, base10 log. Values were normalized by subtracting the means and dividing by the standard deviation. **Gray lines**, overlap between the tBLASTn hit interval and the synder search interval, i.e., tBLASTn finds a hit on the expected strand within the synder-inferred search interval.

4.5 Implementation

`synder` is a C++ program wrapped in an R package via Rcpp [49]. It is designed to be compatible with Bioconductor, an R-based bioinformatics ecosystem [50].

4.6 Availability

As an R package, `synder` should work on any system. It is distributed under a GPL-3 open source license and the source code is available at https://github.com/arendsee/synder. All code required to run the case study is available at https://github.com/arendsee/synder-case-study and the required input data is on DataHub at https://datahub.io/arendsee/synder-nfyc.

Author contributions and acknowledgements

ZA conceived of the project. ZA and AW implemented the program. US and LJ helped with case studies. MH provided extensive feedback on the C++ implementation. ESW provided feedback on the manuscript and algorithms.

Bibliography

[1] Worth, C.L. *et al.* (2009) Structural and functional constraints in the evolution of protein families. *Nature Reviews Molecular Cell Biology* 10, 709

[2] Swint-Kruse, L. (2016) Using evolution to guide protein engineering: the devil is in the details. *Biophysical journal* 111, 10–18

[3] Jain, A. *et al.* (2018) The evolutionary traceability of proteins. *bioRxiv*

[4] Ruiz-Orera, J. *et al.* (2015) Origins of de novo genes in human and chimpanzee. *PLoS genetics* 11, e1005721

[5] Arendsee, Z.W. *et al.* (2014) Coming of age: orphan genes in plants. *Trends in plant science* 19, 698–708

[6] Tautz, D. and Domazet-Lošo, T. (2011) The evolutionary origin of orphan genes. *Nature Reviews Genetics* 12, 692–702

[7] Rivera, M.C. *et al.* (1998) Genomic evidence for two functionally distinct gene classes. *Proceedings of the National Academy of Sciences* 95, 6239–6244

[8] Wall, D. *et al.* (2003) Detecting putative orthologs. *Bioinformatics* 19, 1710–1711

[9] Ohno, S. *et al.* (1968) Evolution from fish to mammals by gene duplication. *Hereditas* 59, 169–187

[10] Zhang, J. (2003) Evolution by gene duplication: An update. *Trends in ecology & evolution* 18, 292–298

[11] Ghiurcuta, C.G. and Moret, B.M. (2014) Evaluating synteny for improved comparative studies. *Bioinformatics* 30, i9–i18

[12] Tang, H. *et al.* (2008) Synteny and collinearity in plant genomes. *Science* 320, 486–488

[13] Cai, B. *et al.* (2011) MicroSyn: A user friendly tool for detection of microsynteny in a gene family. *BMC bioinformatics* 12, 1

[14] Knowles, D.G. and McLysaght, A. (2009) Recent de novo origin of human protein-coding genes. *Genome Research* 19, 1752–1759

[15] Vakirlis, N. and McLysaght, A. (2019) *Computational Prediction of De Novo Emerged Protein-Coding Genes*, pp. 63–81. Springer New York, New York, NY

[16] Ng, M.P. *et al.* (2009) OrthoClusterDB: An online platform for synteny blocks. *BMC bioinformatics* 10, 192

[17] Schwartz, S. *et al.* (2003) Human-mouse alignments with BLASTZ. *Genome research* 13, 103–107

[18] Marçais, G. *et al.* (2018) MUMmer4: A fast and versatile genome alignment system. *PLoS computational biology* 14, e1005944

[19] Grabherr, M.G. *et al.* (2010) Genome-wide synteny through highly sensitive sequence alignment: Satsuma. *Bioinformatics* 26, 1145–1151

[20] Liu, D. *et al.* (2018) Inferring synteny between genome assemblies: a systematic evaluation. *BMC bioinformatics* 19, 26

[21] Larkin, D.M. *et al.* (2009) Breakpoint regions and homologous synteny blocks in chromosomes have different evolutionary histories. *Genome research*

[22] Murphy, W.J. *et al.* (2005) Dynamics of mammalian chromosome evolution inferred from multispecies comparative maps. *Science* 309, 613–617

[23] Cannon, S.B. and Young, N.D. (2003) OrthoParaMap: Distinguishing orthologs from paralogs by integrating comparative genome data and gene phylogenies. *BMC bioinformatics* 4, 35

[24] Begun, D.J. *et al.* (2006) Evidence for de novo evolution of testis-expressed genes in the *Drosophila yakuba/Drosophila erecta* clade. *Genetics* 176, 1131–1137

[25] Toll-Riera, M. *et al.* (2008) Origin of primate orphan genes: a comparative genomics approach. *Molecular biology and evolution* 26, 603–612

[26] Donoghue, M.T. *et al.* (2011) Evolutionary origins of brassicaceae specific genes in *Arabidopsis thaliana*. *BMC evolutionary biology* 11, 47

[27] Langham, R.J. *et al.* (2004) Genomic duplication, fractionation and the origin of regulatory novelty. *Genetics* 166, 935–945

[28] Cui, L. *et al.* (2006) Widespread genome duplications throughout the history of flowering plants. *Genome research* 16, 738–749

[29] Cormen, T.H. *et al.* (2009) *Introduction to algorithms*. MIT press

[30] Mantovani, R. (1999) The molecular biology of the ccaat-binding factor nf-y. *Gene* 239, 15–27

[31] Cicchillitti, L. *et al.* (2017) The laminA/NF-Y protein complex reveals an unknown transcriptional mechanism on cell proliferation. *Oncotarget* 8, 2628

[32] Benatti, P. *et al.* (2016) NF-Y activates genes of metabolic pathways altered in cancer cells. *Oncotarget* 7, 1633

[33] Li, G. *et al.* (2018) The animal nuclear factor Y: an enigmatic and important heterotrimeric transcription factor. *American journal of cancer research* 8, 1106

[34] Nelson, D.E. *et al.* (2007) Plant nuclear factor Y (NF-Y) B subunits confer drought tolerance and lead to improved corn yields on water-limited acres. *Proceedings of the National Academy of Sciences* 104, 16450–16455

[35] Qi, M. *et al.* (2018) QQS orphan gene and its interactor NF-YC 4 reduce susceptibility to pathogens and pests. *Plant biotechnology journal*

[36] Li, L. and Wurtele, E.S. (2015) The QQS orphan gene of Arabidopsis modulates carbon and nitrogen allocation in soybean. *Plant biotechnology journal* 13, 177–187

[37] Swain, S. *et al.* (2017) The multifaceted roles of NUCLEAR FACTOR-Y in *Arabidopsis thaliana* development and stress responses. *Biochimica et Biophysica Acta (BBA)-Gene Regulatory Mechanisms* 1860, 636–644

[38] Gusmaroli, G. *et al.* (2002) Regulation of novel members of the *Arabidopsis thaliana* CCAAT-binding nuclear factor Y subunits. *Gene* 283, 41–48

[39] Lelandais-Briere, C. *et al.* (2016) Noncoding RNAs, emerging regulators in root endosymbioses. *Molecular Plant-Microbe Interactions* 29, 170–180

[40] Combier, J.P. *et al.* (2006) MtHAP2-1 is a key transcriptional regulator of symbiotic nodule development regulated by microRNA169 in *Medicago truncatula*. *Genes & development* 20, 3084–3088

[41] Tang, Y. *et al.* (2017) Arabidopsis NF-YCs mediate the light-controlled hypocotyl elongation via modulating histone acetylation. *Molecular plant* 10, 260–273

[42] Fleming, J.D. *et al.* (2013) NF-Y co-associates with FOS at promoters, enhancers, repetitive elements, and inactive chromatin regions, and is stereo-positioned with growth-controlling transcription factors. *Genome research* pp. gr–148080

[43] Kumimoto, R.W. *et al.* (2010) NF-YC3, NF-YC4 and NF-YC9 are required for CONSTANS-mediated, photoperiod-dependent flowering in *Arabidopsis thaliana*. *The Plant Journal* 63, 379–391

[44] Gnesutta, N. *et al.* (2017) CONSTANS imparts DNA sequence-specificity to the histone-fold NF-YB/NF-YC dimer. *The Plant Cell* pp. tpc–00864

[45] Huang, C.H. *et al.* (2015) Resolution of Brassicaceae phylogeny using nuclear genes uncovers nested radiations and supports convergent morphological evolution. *Molecular biology and evolution* 33, 394–412

[46] Berthelot, C. *et al.* (2014) The rainbow trout genome provides novel insights into evolution after whole-genome duplication in vertebrates. *Nature communications* 5, 3657

[47] Byrne, K.P. and Wolfe, K.H. (2007) Consistent patterns of rate asymmetry and gene loss indicate widespread neofunctionalization of yeast genes after whole-genome duplication. *Genetics* 175, 1341–1350

[48] Innan, H. and Kondrashov, F. (2010) The evolution of gene duplications: classifying and distinguishing between models. *Nature Reviews Genetics* 11, 97

[49] Eddelbuettel, D. *et al.* (2011) Rcpp: Seamless R and C++ integration. *Journal of Statistical Software* 40, 1–18

[50] Gentleman, R.C. *et al.* (2004) Bioconductor: Open software development for computational biology and bioinformatics. *Genome biology* 5, R80

CHAPTER 5. `rmonad`: A MONADIC PIPELINE PROGRAM

Zebulun Arendsee, Jennifer Chang, Eve Wurtele

Modified from a paper submitted to the *R Journal*

Abstract

The `rmonad` package presents a monadic pipeline toolset for chaining functions into stateful, branching pipelines. As functions in the pipeline is run, their results are merged into a graph of all past operations. The resulting structure can be computed on to access not only the final results, but also node documentation, intermediate data, performance stats, and any raised messages, warnings or errors. The structure enables `rmonad` to intercepts all exceptions at any point in the pipeline, which allows for complex pure error handling.

5.1 Background

Pipeline programming is common practice in the R community with `magrittr`, `pipeR`, and `wrapr` packages offering infix pipe operators [1, 2, 3]. The value on the left of the pipe operator is passed as the first argument to the right-hand function. This style of programming simplifies code by removing the need to name intermediate values or write deeply nested function calls. For example, using the `magrittr` pipe operator, `%>%`, the expression `x %>% f %>% g` is equivalent to `g(f(x))`. Thus these pipelines are equivalent to applied function compositions and may be called function *composition* pipelines.

A *monadic* [4, 5] pipeline extends composition pipelines by allowing *context* to be threaded through the pipeline. Each function call in the pipeline produces both a new

value (on successful evaluation) and a computational context surrounding the new value. This new value and context is then merged with the context of the prior node in the pipeline, allowing past context to be stored. The output of a monadic pipeline is not just the value returned by the final function, but also the context. In this way, monadic pipelines can be automatically self-describing by returning both the result and a description of the process that created it.

In this paper, we describe `rmonad`, the first monadic pipeline program developed for the R language. `rmonad` captures the history of a pipeline as a graph of all past operations. Each node in the graph represents either an input or a function. These nodes store the source code, documentation, any raised messages/warnings/errors, benchmarking info, and arbitrary additional metadata. `rmonad` also generalizes the standard linear pipeline to a directed graph with support for branching and looped pipelines.

While *syntactically* `rmonad` resembles function composition pipeline programs, it is *semantically* more similar to the `Make` family of build automation programs, such as GNU Make [6] and the R package `drake` [7]. Like these programs, `rmonad` specifies a graph of dependent operations and is intended to handle large, complex projects. Unlike these programs, `rmonad` builds the graphs incrementally and integrates context with output, wrapping everything into a single R object that can be further computed on after the initial run.

In this paper, we first describe the "Rmonad" object, second describe the `rmonad` implementation of the monadic pipeline operator, third outline the features and usage or `rmonad` and finally tie everything together with a case study.

5.2 The "Rmonad" object and `rmonad` evaluation

The context that is passed through an `rmonad` pipeline is stored as an "Rmonad" S4 object. This object consists of a directed graph of the relationships between nodes in the pipelines, a list containing the information about each node (including the output

if it is cached), and a unique identifier for the *head* node—the node whose output will be passed in the next piping operation. A pipeline consists of a series of expressions that are evaluated using upstream data as an input. Each expression is evaluated by the special `rmonad` function, `as_monad`, that takes an R expression and returns an "Rmonad" object (see **Algorithm 5**). After each new expression in a pipeline is evaluated, the past "Rmonad" object is merged into the new one.

```
1  function as_monad(f, x):
2      metadata <- get_meta(f)
3      doc <- get_doc(f)
4      code <- get_code_string(f)
5      runtime <- time({ result <- pure_eval(f(x)) })
6      isOK <- success(result)
7      if isOK then
8          y <- result$value
9          mem <- size(result$value)
10     end
11     else
12         y <- NULL
13         mem <- 0
14     end
15     return Rmonad(y, isOK, code, metadata, doc, runtime, mem)
```

Algorithm 5: Pseudocode for the `rmonad` eval function, `as_monad`. `get_meta` and `get_doc` are functions that parse the f abstract syntax tree to extract the documentation string and metadata list (if present). `get_code_string` gets the R code of the function as a string. These three functions rely on the metaprogramming aspects of R, which allow functions to operate on the code of their inputs. The `pure_eval` function is like the standard `eval` R function except that it captures error/warning/message output and returns these together with the output value as a list. `$` is used to access a value in a list. `Fsuccess` returns TRUE if the evaluation raised no error. `size` returns the memory footprint of an R object. `Rmonad` is a constructor for an "Rmonad" object. `rmonad_eval` evaluates a function call, captures any raised messages, records information about the function and its output, and returns a new "Rmonad" object.

To introduce `rmonad` evaluation and the concept of pure functions, we will consider the case of the R expression `log(x)`. A *pure function* is a function that uniquely maps from inputs (formal arguments of the function) to an output without causing any side-

effects. Functions in R are generally not pure. For example, whereas the mathematical log function maps from one positive real number to another positive real number (ignoring complex numbers), the R function maps all possible inputs (including negative or non-numbers) to one of the following possible outcomes:

$$\texttt{log(x)} = \begin{cases} \log(x) & \text{if x is numeric and } x > 0 \\ \texttt{-Inf} & \text{if } x = 0 \\ \texttt{NaN} \text{ with warning} & \text{if } x \text{ is numeric and } x < 0 \\ \text{error with message} & \text{if } x \text{ is not numeric} \end{cases} \tag{5.1}$$

The above statement is not a valid mathematical function since it has *effects* (warningsand errors) beyond what is returned by the function. When $x < 0$, the effect is a warning. when x is not numeric (e.g. `log(‘‘a cat’’)`) the effect is an error message and program termination. These effects are part of the larger computational context of the function but are not returned by the function.

The `as_monad` function makes an R expression pure with respect to messages and exceptions by intercepting them and storing them in the returned “Rmonad” object. `as_monad` also records additional information into the “Rmonad” object, including details about the R expression, the execution time, and the size in memory of the output. The *type signature* of `as_monad` is:

$$\text{as\_monad} :: R \to M\ a \tag{5.2}$$

The `as_monad` takes the R expression, R, and returns $M\ a$, which is the “Rmonad” object M wrapping the value returned from the evaluation of R. On success, the returned value has type a. Thus, whereas a composition pipeline would consist of chained functions of type $a \to b$, $b \to c$, $c \to d$, etc, an `rmonad` pipeline consists of $a \to M\ b$, $b \to M\ c$, $c \to M\ d$.

5.3 The monadic pipe operator

Each evaluation step in an `rmonad` pipeline creates a contextualized object. However, including the context in the output causes a type conflict. For example, suppose there are function f and g with types $(a \to M\ b)$ and $(b \to M\ c)$, respectively. The first function produces an output of type $M\ b$, but the second function requires an input of type b. This conflict is resolved through the special composition performed within the monadic pipe operator.

The monadic pipe operator, or the *bind* operator, has the type signature:

$$bind\ ::\ \underbrace{m\ b}_{\text{output of } f} \to \underbrace{(b \to m\ c)}_{\text{the function } g} \to \underbrace{m\ c}_{\text{output of } g} \tag{5.3}$$

Where m is a generic monad. The function `bind` takes an input of type $m\ b$ and the function g of type $(b \to m\ c)$. It returns the output of g which has type $m\ c$. Many functions of the general type $a \to m\ b$ may be chained together using this `bind` function, for example the call `bind(bind(f(x), g),h)` would chain the contextualized results of f through g and then h. Depending on the implementation of the `bind` function, context from $m\ b$ can be passed down the pipeline to $m\ c$.

The simplest possible implementation of the `bind` function passes no state and is identical to applied functional composition (e.g., as done in `magrttr`):

```
1 function bind(x, g):
2 |   y = extract(x)
3 |   z = g(y)
4 |   return z
```

The primary operator of `rmonad`, `%»%`, is similar to `bind` with one subtle difference. The type of the `%»%` operator is

$$\underbrace{M\ a}_{lhs} \to \underbrace{(a \to\ b)}_{rhs} \to \underbrace{M\ b}_{output} \tag{5.4}$$

`%»%` is a binary operator where the left hand side (codelhs) is an "Rmonad" object (M) wrapping a value of type a. The right hand side (`rhs`) is a normal R function, that takes an input of type a and returns a value of type b. If `lhs` stores a failing state (i.e., a prior node in the pipeline raised an error), then the `rhs` function is not evaluated and the failed state is propagated. Otherwise, the value is extracted from `lhs` and `as_monad` then evaluates the `rhs` function with the `lhs` value as its first argument yielding a new "Rmonad" object. Finally, this new object is merged with the prior, `lhs` "Rmonad" object. Merging involves joining the node graphs of the old and new "Rmonad" objects, setting the head of the resulting graph to the head of the new graph, and removing the value stored in the prior head. The "head" of a graph becomes important later when we discuss branching pipelines (see **Algorithm 6**).

The difference between `%»%` and a true monadic `bind` operator is that the `lhs` of a monadic bind operator is a function ($a \rightarrow M\ b$), whereas the `lhs` of `%»%` is a normal R function. The `%»%` adds the additional step of effectively transforming the normal input R function into a function that yields the monadic object. This is carried out within the monadic bind function through the special evaluation offered by `as_monad`.

5.4 Other `rmonad` operators

While the primary `rmonad` operator is the monadic pipe operator, `%»%`, several additional operators are provided for operating on "Rmonad" objects using pipeline syntax (listed in **Table 5.1**). The `%*>%` operator takes a list of "Rmonad" objects on the left and feeds the values of each as arguments into the function on the right, linking the history of each input "Rmonad" object to the final "Rmonad" object. This operator is important in building branching pipelines. The `%__%` operator is like a semicolon in a programming language, separating independent pipelines but passing on context. In `rmonad`, the `%||%` and `%|>%`, operators are used in error recovery.

```
1  function rmonad_bind(lhs, rhs):
2      h <- head(lhs)
3      if failed(h) then
4          return lhs
5      end
6      else
7          r2 <- as_monad(rhs, value(h))
8          r3 <- union(lhs, r2)
9          if failed(r2) then
10             r3 <- set_value(r3, value(h))
11         end
12         return r3
13     end
```

Algorithm 6: The `%»%` `bind` function. `lhs` and `rhs` are the left hand side and right hand side of the binary `%»%` operator, respectively. `lhs` is an "Rmonad" object, which is a graph of past operations and stored information about them. `head` extracts the current node in the graph that is being acted on (the "Rmonad" object stores the index of the current head). `failed` returns TRUE if the operation stored in its argument raised an error. `value` returns the data stored in a node (or in the head node of an "Rmonad" object). `rmonad_eval` evaluates an R function and its arguments and returns a singleton "Rmonad" object (see **Algorithm 5**). `union` merges two "Rmonad" objects, retaining the head of the second object. Here the second "Rmonad" object is a singleton, so we are adding one node to the function graph and making it the new head node. `set_value` sets the value of the head node in an "Rmonad" object. `rmonad_bind` returns a new "Rmonad" object with a new value on success and the old value on failure.

Operator	Description
`%»%`	pass `lhs` as initial argument of `rhs` function
`%v>%`	like `%»%` but caches the `lhs` value
`%*>%`	pass list of arguments from `lhs` to `rhs`
`%__%`	`rhs` starts a new chain that preserves `lhs` history
`%||%`	use `rhs` value if `lhs` is failing
`%|>%`	call `rhs` on `lhs` if `lhs` failed

Table 5.1 A partial list of the supported operators. `lhs` and `rhs` refer to the left–hand and right-hand sides of the given binary operator. `%»%` is the primary monadic chain operator. The last two operators are for exception handling and traceback.

5.5 Exception handling and tracebacks

The core functionality of `rmonad` is the stateful data piping provided by the monadic operator `%»%`. Linear chains of operations can be constructed with this operator, where each successful node stores information about the function and results. In the case of an error, `rmonad` provides access to both the traceback and the inputs to each failing function. Knowing the error messages and the function inputs allows the programmer to step through the failed function and easily diagnose the problem. All information is stored within the "Rmonad" object, rather than in the ephemeral state of an R session.

Here is a concrete example:

```
m <- "a cat" %>>% log %>>% sqrt
get_error(m)
#> [[1]]
#> character(0)
#>
#> [[2]]
#> [1] "non-numeric argument to mathematical function"
get_code(m)[[2]]
#> "log"
get_value(m)[[2]]
#> [1] "a cat"
```

Here an illegal value is passed into the natural log function. `rmonad` catches this error and saves the final failing input and error message.

5.6 Branching: generalizing the linear pipeline to a directed graph

An "Rmonad" object stores a directed graph of all past operations. Any of these operations may be tagged, cached, and reused in a future operation. A "Rmonad" object specifies a special "head" node in the graph whose value will be passed when the graph is chained into the next function via `%»%`. There are two major ways to create branches: 1) the head can be reset to an internal node that can later serve as branching pipeline, and

2) the `%*>%` operator can call a right-hand side function on multiple "Rmonad" internal nodes.

Since `rmonad` pipelines are branched, there is in general no single output value of the pipeline. Rather, the data contained in the "Rmonad" object is queried using a family of vectorized getter functions. For example, `get_value` will return a list containing the value stored in each node (or `NULL` if no value is stored); `get_error` returns all error messages, `get_warning` returns all warnings, `get_code` returns all code strings, etc. These getter functions can all be parameterized with the node indices or tags.

```
m <- "a" %>>% paste("cat") %>>% sqrt
get_code(m, index=!get_OK(m))
```

One node may have multiple tags. This is particularly useful for accessing all elements in a loop:

```
m <- loop(
  as_monad(letters[1:3]),
  function(x){ x %>>% paste("!") %>% tag(c("letters", x)) }
) %*>% paste
get_value(m, tag="letters")
#> $`letters/c`
#> [1] "c !"
#>
#> $`letters/b`
#> [1] "b !"
#>
#> $`letters/a`
#> [1] "a !"
```

5.7 Nesting: integrating small pipelines to build complexity

`rmonad` has support for nested pipelines **Figure 5.2**. `rmonad` parses the nested function to find connections between the inputs to the nest parents and the nodes in the nested pipelines that they connect to. These connections allow the `rmonad` graph to represent the correct relationships between nested nodes and their inputs, and this, for

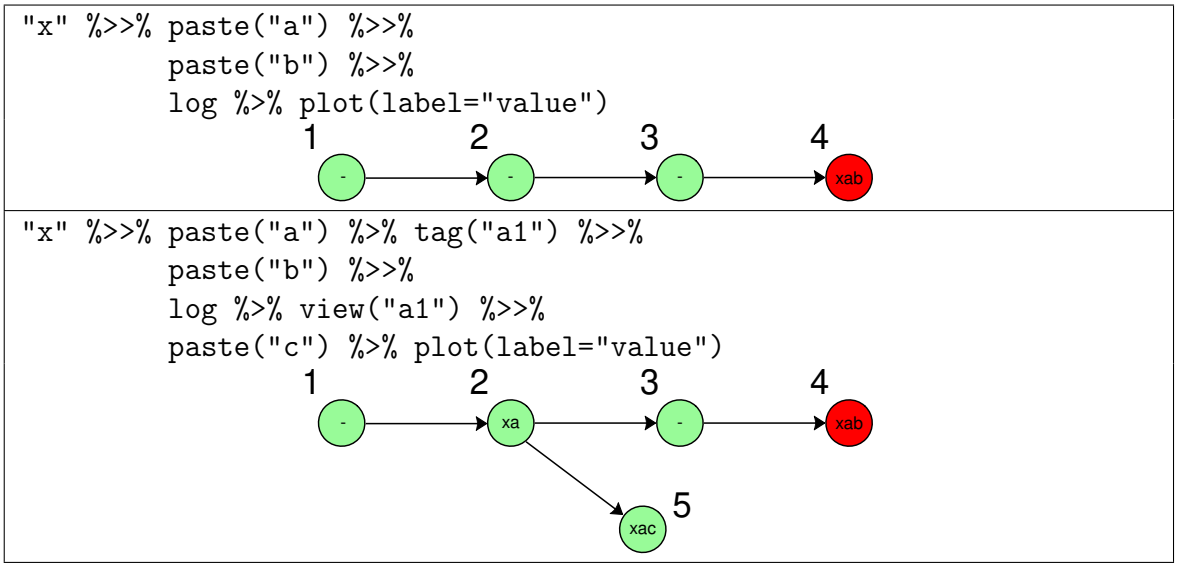


Figure 5.1 `rmonad` code examples and plots. The final `plot` function creates the graph with values in nodes if the values are cached and "-" otherwise. The plots were modified in the vector editor Inkscape. **Top:** A linear `rmonad` pipeline that ends in an error and its resulting graph. The pipeline begins at node 1 with the value "x". This is piped into the `paste` function which concatenates the letter "a". Since the `paste` is successful, the result is stored in node 2 and the value in node 1 is deleted to save memory. The value in node 2 is piped into `paste` again, concatenating the letter "b" and storing it in node 3. The value in node 3 is piped into the `log` function, where an error is raised, terminating this branch, and storing the final failing value, "x a b", and the error message. The value is only stored at the end node so we do not end up with duplicate values all the way down a pipeline. That way, values are stored when there are errors, or where explicitly tagged by the user—for example for branching. **Bottom:** A branched `rmonad` pipeline and its resulting graph. From node 2, the "Rmonad" object is piped into the `tag` function which annotates the head node (node 2) with the tag "a1" and sets a flag that ensures the value will be cached for later use. After function 4, the "Rmonad" object is piped into `view` which sets the head of the graph to node 2. Finally, the value in node 2 is piped into the final `paste` function that concatenates "c".

example, allows the nested nodes to store the correct inputs if they fail. Thus `rmonad` allows for multilevel debugging. While tracebacks of errors is commonly available, `rmonad` also stores the input to each failed function at each nest level. This allows the programmer to step through the code in the failed node using the input data, without having to rerun the pipeline. Nesting also allows abstraction by building complex pipelines from smaller pipelines wrapped in functions.

```
# Level 2
f <- function(x) {
  "<" %v>% paste(x) %v>% paste(">")
}
# Level 1
"A" %v>% paste("B")
    %v>% f
    %v>% paste("C") %>% plot(label="value")
```

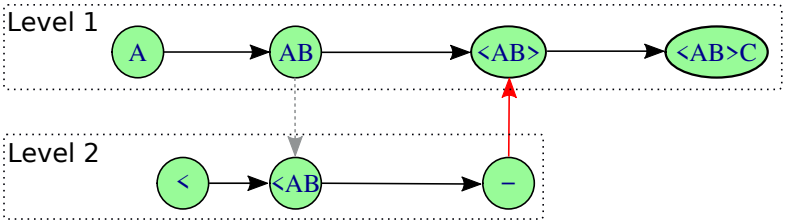


Figure 5.2 Nesting Pipelines. Level 1 is a pipeline where the 3rd node wraps the pipeline in Level 2. Arrows show relationships. The **black** arrow is data being passed directly to a new function. The **gray** arrows shows data being passed into the nested function from one of the inputs to the node that nests the function. The **red** arrows show that the terminal node in the nested pipeline passes its value to the parent node.

5.8 Parsing code strings, docstrings and metadata lists

`rmonad` leverages R non-standard evaluation to parse the abstract syntax tree of functions at runtime prior to their evaluation. `rmonad` extracts 1) the function's code as a string, 2) a documentation string, and 3) a list of arbitrary metadata. All three items are stored in the "Rmonad" node. For example:

```
foo <- function(x){
  "This is a docstring"
  list(sysinfo = sessionInfo())
  return(x)
}
```

The first two lines in the function body are the docstring and metadata list, respectively. Each must 1) be of the appropriate type (string and list, respectively), 2) not be assigned to a variable, and 3) not be the final line in the function body. `foo` is a legal R function that can be used naturally outside of the `rmonad` context. The docstring and metadata would be "dead" lines of code that are evaluated but that are not assigned to any variable or returned. When `rmonad` parses the function before evaluation, the first two lines will be removed and stored, yielding the following function for evaluation:

```
function(x){
  return(x)
}
```

The docstring and the function code are stored as simple strings. The metadata list is evaluated within the function environment, giving it access to function input, and then stored.

The metadata is an arbitrary list associated with a node. This could be used to store static data such as the author's name, a version for the function, arbitrary notes. It can also store report generation parameters (like code chunks in `knitr`). Since the list is evaluated, the contents are dynamic, allowing, for example, session info to be stored or `knitr` parameters to be a function of the input. Whereas `knitr` nests code chunks and their parameters in a text document, `rmonad` nests text and parameters within the code.

The metadata can be modified freely even *after* the pipeline is run, to enable the user to store notes that are a function of the pipeline results, or personal annotations, reminders, and comments on the results.

5.9 Post-processing: formatting, summarizing, and logging

A built-in use of the metadata is to add formatters, summarizers, and loggers that are executed automatically after a node is run. For example, a pipeline developer might write the function:

```
fancy_log <- function(x){
  list(
    format_warnings = function(x, xs) {
      sprintf("%s NaNs produced", sum(is.na(x)))
    },
    log = function(x, passing) {
      if(passing){
        cat("pass\n")
      } else {
        cat("fail\n")
      }
    },
    summarize = list(len = length)
  )
  log(x)
}
```

When run, the captured warnings are processed by the `format_warnings` function, with the following result:

```
"a cat" %>>% fancy_log -> m
#> fail
c(-2,-1,0,1,2) %>>% fancy_log -> m
#> pass
get_warnings(m)
#> [[1]]
#> character(0)
#>
#> [[2]]
#> [1] "2 NaNs produced"
> get_summary(m)[[2]]$len
#> 5
```

In the first case, an illegal value is passed to the `fancy_log` function. This leads to a failure in the second node, and the logger prints "fail". In the second case, the user

passes the integers between -2 and 2, storing the result in `m`. Since these are legal values (from R's perspective), the logger prints the message "pass" after evaluation. When the returned object is printed, the post-processed warning message "2 NaNs produced" is shown. The result of the summarizing function is accessed through the `get_summary` function.

5.10 Case Study

As an example of a branching `rmonad` pipeline with error, warning and run time handling; we analyzed the iris dataset [8, 9] using three different statistical methods: (1) ANOVA, (2) Kruskal-Wallis, and (3) t-test. The Iris dataset is an often-used dataset for statistics and machine learning, and consists of features of three species of flowers: *Iris setosa*, *Iris virginia*, and *Iris versicolor*. Among these features is petal length. We used the three statistical methods to determine if petal length is significantly different across the three *Iris* flower species. Some statistical methods are not appropriate for the dataset without data pre-processing. This case study provides an example of running multiple methods using a branching `rmonad` pipeline, while comparing the output and running times of each method.

Normally, programmers would run the three methods separately using an R script similar to the following:

```
data(iris)

# === 3 Statistical Tests (run one at a time)
# (1) Anova
res.aov <- aov(Petal.Length ~ Species, data = iris)
summary(res.aov)

# (2) Kruskal-Wallis
res.kr <- kruskal.test(Petal.Length ~ Species, data = iris)
res.kr

# (3) T-Test
t.test(Petal.Length~Species, data=iris)
```

Using `rmonad` tags, data can be branched out to encompass the three statistical tests. Here, the R variable m stores the output "Rmonad" S4 object. We must initially tag the branch point node (in this case the original Iris dataset). Since the first node has been given a tag ("indata"), its value will be cached and can be accessed with the command `get_value(m, tag="indata")`. From here, we can access and pipe (`%»%`) the viewed "indata" tag into the different statistical tests, as scripted below and visualized in **Figure 2**.

```
# === rmonad (run together)
m <- {
  "iris dataset"
  as_monad(iris, tag="indata")
} %>>% {
  "anova"
  res.aov <- aov(Petal.Length ~ Species, data = .)
  summary(res.aov)
}

m <- {
  view(m, "indata")
} %>>% {
  "Kruskal-Wallis"
  res.kr <- kruskal.test(Petal.Length ~ Species, data = iris)
  res.kr
}

m <- {
  view(m, "indata")
} %>>% {
  "t-test"
  t.test(Petal.Length~Species, data=iris)
}
```

The above code could have been chained together using `%>% get_value(tag="indata") %>%` commands but were separately added to the m rmonad object for ease of reading. From the m rmonad object, we can plot the pipeline. In the following command we label the nodes by node id, documentation, running time, and any errors if they exists.

```
plot(m, label = function(m){paste(get_id(m),
                                  get_doc(m),
                                  get_time(m),
                                  gsub("character\\(0\\)", "", get_error(m)),
                                  sep=":")})
```

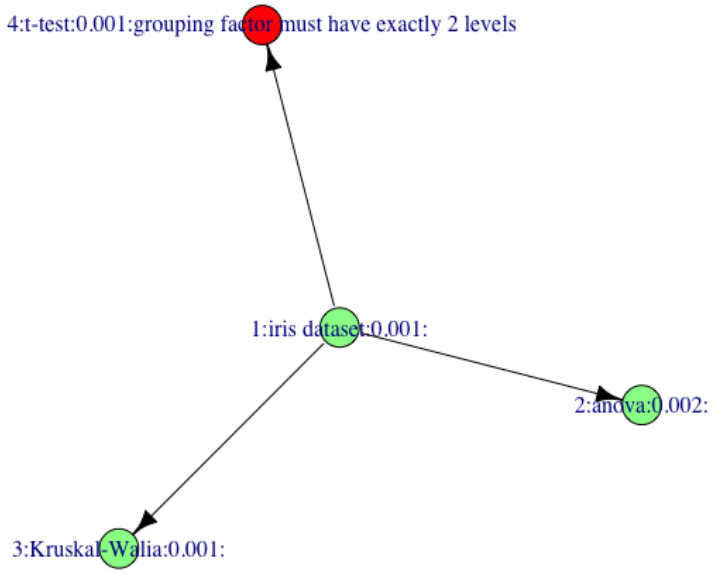


Figure 5.3 Three Statistical Tests. The iris dataset is piped to (1) ANOVA, (2) Kruskul-Wallis, and (3) t-test. Green and red determine if the test ran or threw an error. Time in seconds shown next to the name. Any errors are annotated on the node. Notice how t-test has a "grouping factor must have exactly 2-levels" error. Also, of the two tests without errors, ANOVA ran slightly slower than Kruskal-Wallis.

In **Figure 5.3**, the center node is the iris dataset and has three arrows going outwards toward one red and two green nodes. Of those, the red node near the top represents the t-test and shows the expected error "grouping factor must have exactly 2 levels". Since we are testing the petal length among the three species, this error is expected. Any errors of the pipeline can also be reported in a table:

```
missues(m)
#>   id  type                                          issue
#> 1  4 error grouping factor must have exactly 2 levels
```

Going clockwise, ANOVA and Kruskal-Wallis are represented by nodes 2 and 3. The green nodes indicate that both ran although their running times were different. From

their node labels, Kruskal-Wallis ran in 0.001 ms, slightly faster than ANOVA (0.002). Also note that green nodes only indicate that the method ran successfully, not the results of that method or statistical significance. The results of the ANOVA and Kruskal-Wallis test can be pulled out of the pipeline using their Node ID number and the following commands.

```
> id=c(2,3)           # place id(s) of end result(s) here
> get_value(m)[id]
#> [[1]]
#>              Df Sum Sq Mean Sq F value Pr(>F)
#> Species       2  437.1  218.55    1180 <2e-16 ***
#> Residuals   147   27.2    0.19
#> ---
#> Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#>
#> [[2]]
#>  Kruskal-Wallis rank sum test
#>
#> data:  Petal.Length by Species
#> Kruskal-Wallis chi-squared = 130.41, df = 2, p-value < 2.2e-16
```

Both tests agree that there is a significant difference between `Petal.Length` across the three Iris species. ANOVA ran on the dataset, which means that petal length follows a normal distribution within each species. Kruskal-Wallis does not assume a normal distribution. The analyst can decide which method to use; in this case the conclusion is the same. **Figure 5.3** is an example of a branched `rmonad` pipeline comparing three different statistical methods applied to the iris dataset to test a hypothesis.

5.11 Conclusion

We have presented a method to incorporate monadic pipelines into R using the `rmonad` package. `rmonad` provides an infrastructure for data analysis and report generation. `rmonad` stores pipeline results and metadata that can be easily explored interactively and collated into reports using tools such as the literate programming package `knitr` [10] or the HTML report generator `Nozzle.R1` [11].

`rmonad` integrates a simple profiler into the workflows by automatically capturing the runtime and memory usage of each node. This feature makes it easier for the pipeline developer to determine bottlenecks in the code or potential culprits of memory overflow. Often, a coder must add benchmarking code in a pipeline, like an engineer using a voltmeter testing an electric current in a few key places at a time. `rmonad` has benchmarking built in so all locations are automatically tested and performance can be checked post-run.

`rmonad` will also be useful as a tool for creating and resolving issue reports. If an `rmonad` pipeline fails, the resulting object will store all failing functions, their raised error/warning messages and also their inputs. This object can be sent to a pipeline maintainer, who can then find the error messages, load all inputs to the failing function, and then proceed to step through the code until the bug is found. By prepending a node that stores the local session data (`sessionInfo() %__% ...`) the debugger also gains access to all the user machine's state (an often requested item in a bug report). An "Rmonad" object with session info attached in this way contains everything that the maintainer needs to handle the issue. This could streamline issue resolution by allowing better automation and simplifying the submission process.

Overall, `rmonad` integrates the concepts of a pipeline, a build system, a data structure, and an low-level report generating engine. An `rmonad` project is built up through incremental piped operations (like a pipeline program), supports complex branching projects (like a build system), and produces a data structure that can be computed on to generate dynamic reports.

Currently `rmonad` focuses on handling the complexity of workflows. A future goal is to add automatic parallelism and cache handling. This paper was motivated by the desire to publish best practices in pipelines in R and to encourage better pipeline design and maintenance.

5.12 Availability

`rmonad` is published under the GPL-3 license and is available on the Comprehensive R Archive Network (CRAN) and on GitHub at https://github.com/arendsee/rmonad. Systematic documentation of the features with simple examples can be found in the vignettes, available through CRAN.

Author Contributions

ZA developed `rmonad`. ZA and JC wrote the manuscript. JC made the case study.

Bibliography

[1] Bache, S.M. and Wickham, H. (2014) *magrittr: A Forward-Pipe Operator for R*. R package version 1.5

[2] Ren, K. (2016) *pipeR: Multi-paradigm pipeline implementation*. R package version 0.6.1.3

[3] Mount, J. and Zumel, N. (2018) Dot-Pipe: an S3 extensible pipe for R. *The R Journal*

[4] Eilenberg, S. *et al.* (1965) Adjoint functors and triples. *Illinois Journal of Mathematics* 9, 381–398

[5] Barr, M. and Wells, C. (1985) *Toposes, triples and theories*, vol. 278. Springer-Verlag New York

[6] Stallman, R.M. *et al.* (2002) *GNU Make: A program for directed compilation*. Free software foundation

[7] Landau, W.M. (2017) *drake: Data frames in R for Make*. R package version 4.4.0

[8] Anderson, E. (1936) The species problem in iris. *Annals of the Missouri Botanical Garden* 23, 457–509

[9] Fisher, R.A. (1936) The use of multiple measurements in taxonomic problems. *Annals of eugenics* 7, 179–188

[10] Xie, Y. (2015) *Dynamic documents with R and knitr*. Chapman and Hall/CRC, 2nd edn. ISBN 978-1498716963

[11] Gehlenborg, N. *et al.* (2013) Nozzle: A report generation toolkit for data analysis pipelines. *Bioinformatics* 29, 1089–1091

CHAPTER 6. `fagin`: SYNTENY-BASED PHYLOSTRATIGRAPHY AND FINER CLASSIFICATION OF YOUNG GENES

Zebulun Arendsee, Jing Li, Urminder Singh, Priyanka Bhandary, Arun Seetharam, Eve Syrkin Wurtele

Modified from a paper submitted to *BMC Bioinformatics*

Abstract

Background: With every new genome that is sequenced, thousands of species-specific genes (orphans) are found, some originating from ultra-rapid mutations of existing genes, many others originating *de novo* from non-genic regions of the genome. If some of these genes survive across speciations, then extant organisms will contain a patchwork of genes whose ancestors first appeared at different times. Standard phylostratigraphy, the technique of partitioning genes by their age, is based solely on protein similarity algorithms. However, this approach relies on negative evidence — a failure to detect a homolog of a query gene. An alternative approach is to limit the search for homologs to syntenic regions. Then, genes can be positively identified as *de novo* orphans by tracing them to non-coding sequences in related species. **Results:** We have developed `fagin`, a synteny-based pipeline in the R framework. `fagin` determines the genomic context of each query gene in a focal species compared to homologous sequence in target species. We tested the `fagin` pipeline on two focal species, *Arabidopsis thaliana* (plus four target species in Brassicaseae) and *Saccharomyces cerevisiae* (plus six target species in Saccharomyces). Using microsynteny maps, `fagin` classified the homology relationship of each query gene against each target genome into three main classes, and further

subclasses: *AAic* (has a coding syntenic homolog), *NTic* (has a non-coding syntenic homolog), and *Unknown* (has no detected syntenic homolog). `fagin` inferred over half the *"Unknown" A. thaliana* query genes, and about 20% for *S. cerevisiae*, as lacking a syntenic homolog because of local indels or scrambled synteny. **Conclusions:** `fagin` augments standard phylostratigraphy, and extends synteny-based phylostratigraphy with an automated, customizable, and detailed contextual analysis. By comparing synteny-based phylostrata to standard phylostrata, `fagin` systematically identifies those orphans and lineage-specific genes that are well-supported to have originated *de novo*. It flags genes that may have originated through rapid divergence. `fagin` also delineates whether a gene has no syntenic homolog because of technical or biological reasons, and indicates that some orphans may be associated with regions of high genomic perturbation.

6.1 Background

One of the surprises of the genomic era was that gene birth is not a dead process. The prior paradigm that proteins evolve only by gradual "tinkering" with existing material [1] was contradicted when the sequencing of the first genomes uncovered many species-specific "orphan" genes [2]. Most researchers argued then that the uniqueness of these genes was an artifact of sparse sampling or bad gene prediction, and that when enough genomes were sequenced, all correctly annotated genes would cluster into large, ancient families. But more sequencing proved exactly the opposite. Researchers have shown that not only can genes encoding novel proteins arise *de novo* [3, 2], but they do so often, as shown, for example, in animals [4, 5, 6], plants [7], protists [8], and yeast [9]. In addition to *de novo* orphan genes, other orphan genes encoding novel proteins can be derived from the very rapid mutation of existing CDSs beyond recognition [10]. Orphan genes exist, but what roles do they play?

Although most of the approximately several billion orphan genes [11] have never been studied, functions are being shown for a growing minority. The emerging theory is that

that young genes are common in arenas where fitness optima change quickly, such as environmental response and inter-species relations. Studies have consistently shown that orphans are over-represented among genes that respond to stress [12, 13, 14, 15]. They may also be major contributors to taxonomically-restricted traits [16, 17]. Orphans also may play important roles in developmental cascades [18]. Other orphans are crucial to interspecies conflicts [19], self incompatibility [20], host-pathogen relations [21], and symbiosis [22, 23]. One of the best-studied orphan genes, QQS of *Arabidopsis thaliana*, responds to biotic stresses by altering carbon and nitrogen partitioning [15, 24] and by conferring broad-spectrum pest and pathogen resistance [12]. A study of three *de novo* genes in mice, randomly selected from among very young genes that were inferred to be of *de novo* origin, found evidence of associated phenotypes (longer limbs, changed behavior, and slower life history) [25].

In addition to studies revealing the function of individual orphan genes, there is experimental evidence that functional, beneficial proteins can be produced from random sequence. First, *in vitro* protein evolution from random protein libraries demonstrates that functional proteins can be produced through chance mechanisms [26, 27, 28, 29]. Second, expression of randomly-generated ORFs *in vivo* can lead to phenotypic consequences. About 50% of random ORFs expressed in *E. coli* inhibited growth rate, while about 25% increased growth rate [30]. Of 2000 *A. thaliana* plants expressing random ORFs, ten biologically-relevant phenotypes were revealed and experimentally verified, including early flowering and red light insensitivity [31].

If new genes can arise *de novo*, and thus new genes are constantly appearing, then some should survive across speciation events. Thus, genes in extant species should be stratified into sets of genes that appeared at different times. The technique of inferring the evolutionary time of origin of each gene across a genome is known as phylostratigraphy [32]. Phylostratigraphy is the study of the distribution of gene birth events across deep time by stratifying modern genes by age. In standard phylostratigraphy, the phy-

lostratum of a given protein-coding gene is based on the age of the oldest clade that contains its inferred protein-coding homolog (e.g., [33]). Phylostratigraphy has been used to link clusters of clade-specific genes to the origins of clade-specific traits, such as brain development [33] or the early origins of cancer genes [34]. It also offers snapshots of proteins of different ages and thus provides a unique window into protein evolution, offering insight into the evolution of novel biological features [16].

Standard phylostratigraphic classification based on protein similarity alone has several challenges. A much debated limitation is the difficulty of distinguishing orphan homologs of small, rapidly evolving genes from orphans of *de novo* origin [35, 36]. Another limitation is that phylostratigraphy infers gene ages based on negative evidence: the absence of a detectable, annotated, protein-coding homolog outside a clade. Thus, standard phylostratigraphy does not distinguish genes that are true orphans from those that are missing in related species due to bad genome assemblies or incorrect gene models.

An alternative approach to establish the *de novo* origin of a gene is to search for positive evidence of non-coding sequence in close relatives of the focal species. While in principle, this could be accomplished by simply searching the nucleotide sequence of the focal gene against whole genomes of related species, the large size of a genome and the often low-complexity of the novel gene, make false positives likely. A more powerful technique is to leverage syntenic data to identify the regions in the target genome where a homolog to each focal gene is expected to reside [37, 7]. By searching just this small region, the confidence that a similar sequence represents an ortholog is improved.

Syntenic analysis has provided a powerful approach to distinguish young genes with a *de novo* origin from genes encoding proteins which are unrecognizable in closely related species because they have undergone rapid evolutionary change [37, 7]. However, the use of synteny has been mostly limited to specialized, study-specific analyses [37] or to cases where tools are available for curated selections of genomes, such as the UCSC genome

browser [25, 38]. Until now, no general genome-wide solution has been available for synteny-informed phylostratigraphy analysis.

Here, we present `fagin`, a new R package that generalizes, refines, and automates syntenic phylostratigraphy synteny-based phylostratigraphy. `fagin` facilitates comparative analysis of genes across evolutionary clades, augmenting standard phylostratigraphy with a detailed, synteny-based analysis. Whereas standard phylostratigraphy searches the proteomes of related species for similarities to focal genes, `fagin` first finds syntenic genomic intervals and then searches within these intervals for any trace of similarity. It searches the (*in silico* translated) amino acid sequence of all unannotated ORFs as well as all known CDS within the syntenic search space of the target genomes. If no amino acid similarity is found within the syntenic search space, `fagin` will search for nucleotide similarity. Finding nucleotide sequence similarity, but not amino acid similarity, is consistent with a *de novo* origin of the focal gene. If no similarity of any sort is found, `fagin` will use the syntenic data to infer a possible reason. For example, `fagin` can detect indels, scrambled synteny, assembly issues, and regions of uncertain synteny.

`fagin` makes three major contributions to the phylostratigraphy field. 1) *Automation.* `fagin` offers the first automated, package for synteny-based phylostratigraphy. 2) *Fine-tuned classification of query gene homologies.* By dividing homology inferences into three general classes (amino acid, nucleotide, and unknown), each with a set of subclasses, rather than using the typical binary classification (amino acid or nucleotide) for syntenic analysis, `fagin` provides a basis for assessing confidence in phylostratigraphy classifications and *de novo* designations. This makes `fagin` robust against bad data: genes in regions that are poorly assembled will fall into one of the Unknown-technical classifications. Also, if gene annotations are missing, matches against ORFs in the syntenic regions of the target genome will still be found (some of these matches may represent genes that are unannotated in the target genome; others may represent very rapidly-changing genes). 3) *Flexibility in (micro)synteny maps.* Whereas prior syntenic

studies have been limited to synteny maps based on orthologous genes [37, 39], `fagin` can handle any synteny map, and is indeed particularly suited to micro-synteny maps produced by whole genome alignments. These fine-grained maps allow higher resolution through smaller inferred search intervals. They are also the basis for the inferred subclassifications.

As proof-of-concept, we explore the use of `fagin` in two cases studies centered on the focal species *Saccharomyces cerevisiae* and *Arabidopsis thaliana*. We systematically identify genes that have arisen *de novo* from non-coding precursors and rapidly evolving genes that may have been missed by more traditional methods of gene annotation.

6.2 Implementation

The `fagin` pipeline can be sub-divided into three stages (**Figure 6.1**): 1) process input data; 2) search syntenic regions on target genomes for sequence similarity to query genes; 3) infer gene origins by comparing across genomes of related species. The entire pipeline is built using the `rmonad` pipeline package (available on CRAN). `rmonad` is designed to simplify the documentation, organization, benchmarking, and debugging of complex data analysis pipelines.

6.2.1 Required input data

The inputs required for `fagin` are: 1) a phylogenetic tree relating the focal species to one or more target species; 2) a genome sequence for the focal species and each target species; 3) Genome Feature Format (GFF) files that describes all gene models (or other features of interest) for each species; 4) the genes (or other features) to be queried from the focal species; and 5) pairwise synteny maps between the focal species genome and the genomes of each target species. The synteny maps are constructed from the genome pairs using an outside program. For our case studies we used `MUMmer4` [41] (for *S. cerevisiae*

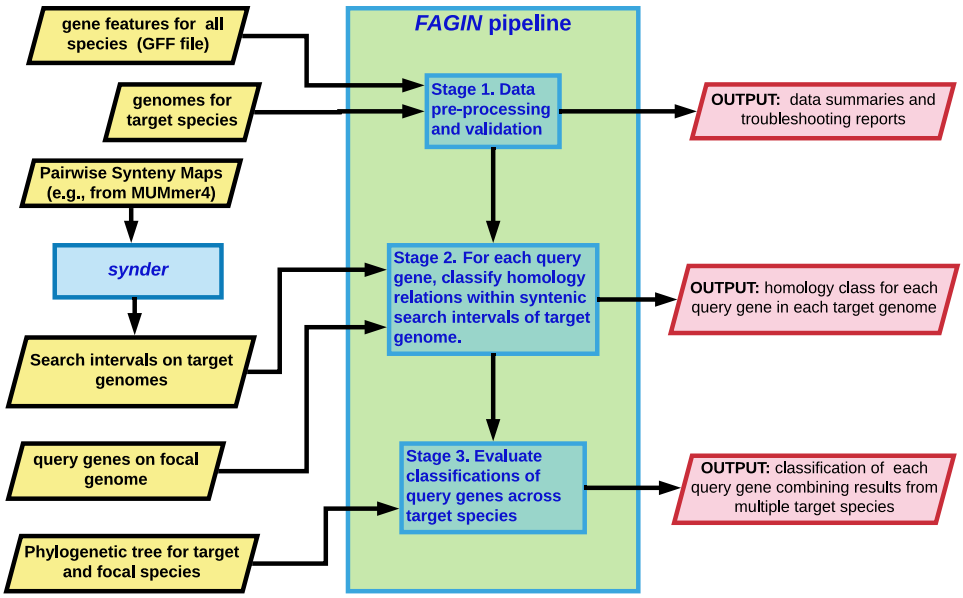


Figure 6.1 Overview of the `fagin` pipeline. **Inputs** (yellow rhombuses) are passed into `fagin`; the syntenic search intervals on the target genome corresponding to each query gene are delineated using `synder` [40]. The `fagin` pipeline consists of three stages. **Stage 1:** all input is validated, summarized, and secondary data (protein sequences, transcripts, ORFs) are extracted from genomes. **Stage 2:** The search intervals in the genomes of the target species that correspond to each query genes are searched to determine whether there is homology to the amino acid sequence ($AAic$) or nucleotide sequence ($NTic$) of that query gene, and if so where the homology occurs. Alternately, no homology might be detected (*unknown*). **Stage 3:** For each query gene, the homology classes are compared across the phylogenetic tree to infer that gene's history. `fagin` is customizable by the user. **Output** (red rhombuses) can include, e.g., summaries of the transformed input data, homology classes for each query gene against each target genome with statistical designations, and summaries of the homology results for each query gene across all genomes.

against other species in the genus) and Satsuma [42] (for *A. thaliana* against Brassicaceae relatives).

6.2.2 Stage 1: process input data and infer syntenic search intervals

In *Stage 1*, `fagin` cleans, validates, and summarizes all of the input data. The format of all input files is checked by `fagin`. Then, from the GFF files and genome sequences, `fagin` derives the protein sequences, transcript sequences, coding sequences (CDS), and the open reading frames (ORFs) on transcripts and whole genomes (see **Section A.4**). The most difficult data processing step is extracting gene models from the GFF files (see **Section A.3** for details). `fagin` also checks for signs of invalid input, such as stop codons appearing in the derived protein sequences. Then, `fagin` summarizes the assemblies and annotations of all genomes, the derived protein sequences, and the synteny maps.

`fagin` also infers syntenic search intervals for each focal gene on each target genome, using input from the `synder` package [40]. `synder` traces each query gene on the focal species to a *search space* on the target genome, which is a set of one or more genomic intervals that are inferred to be orthologous. The purpose of delineating search intervals is to winnow false positives and increase sensitivity by limiting the search to orthologous regions of the target genome. In Stage 2, these syntenic search intervals are analyzed to find traces of homology to the CDS of the query gene.

6.2.3 Stage 2: determine homology classes of each query gene in the search interval of each target genome

In *Stage 2*, each query gene is assigned, relative to its inferred search intervals, to a homology class (**Figure 6.2**). By default, `fagin` considers three general cases (**Figure 6.3**): **AAic** if there is aa similarity between the protein encoded by the query gene and the translation product of a known CDS or any other ORF within the search interval; **NTic**, if there is nucleotide similarity of the query gene to any nucleotide sequence (transcript or genomic) within the search interval; and **Unknown** if no similarity can be

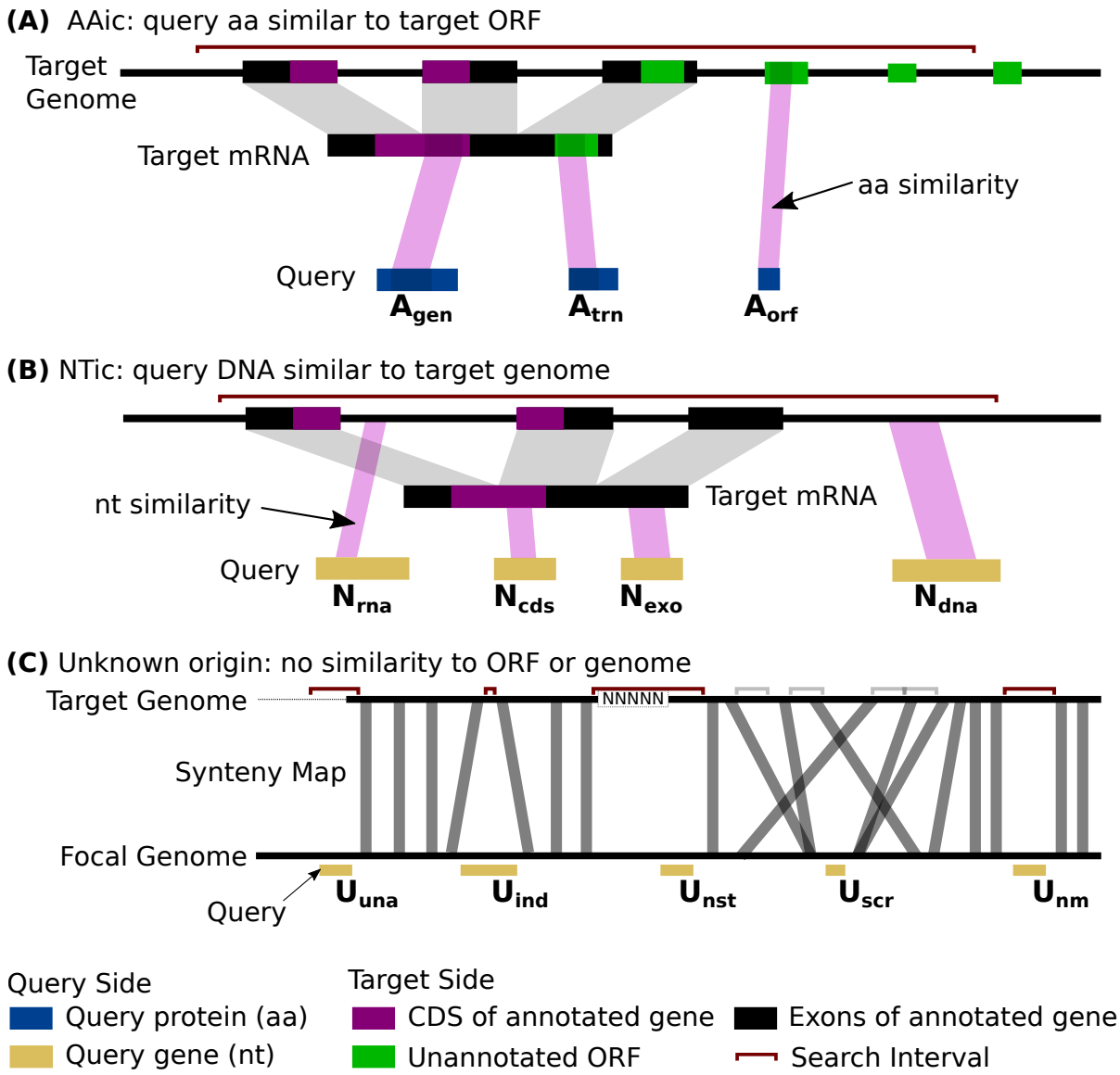
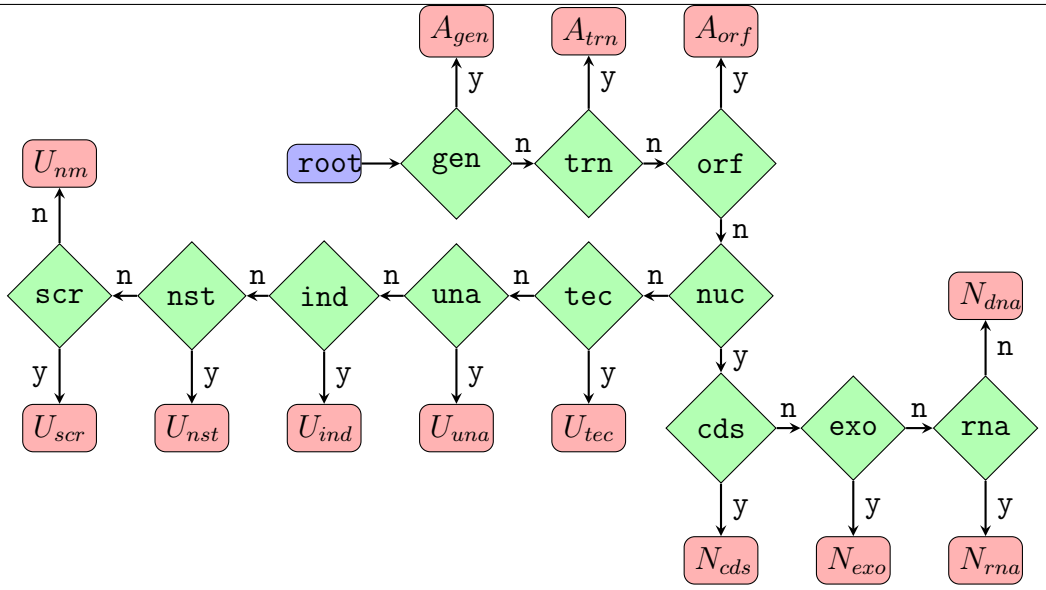


Figure 6.2 `fagin` classifies the genomic context. `fagin` infers genomic context of query genes or other genomic features on the focal genome by searching for homologous sequence within syntenic search intervals on the target genome. For protein-coding query genes, `fagin` searches for homology to the protein (aa)(**A**) or entire sequence (nt) (**B**) of the query gene. It also categorizes the unknown (**C**). Grey bars in **C**, syntenic links. The `fagin` classification is indicated below each queries, in bold black font. Rooting the homology searches to the syntenic regions narrows the search space, thereby increasing the sensitivity.



node	homology class
`gen`	A_{gen} if there is a significant[1] aa match to a known[2] protein
`trn`	A_{trn} if there is a significant[1] aa match to a translated ORF[3] on a known mRNA[2]
`orf`	A_{orf} if there is a significant[1] aa match to any translated ORF[3]
`nuc`	NT match; has significant[1] DNA match somewhere in the search interval
`cds`	N_{cds} if the DNA match overlaps a CDS[2]
`exo`	N_{exo} if the DNA match overlaps an exon[2]
`rna`	N_{rna} if the DNA match overlaps an intron[2] and N_{dna} if the DNA match does not overlap a transcript of a known gene
`tec`	U_{tec} if the search space is too big for `fagin` to search
`una`	U_{una} if the search interval maps to the end of a scaffold (incomplete assembly)
`ind`	U_{ind} if search interval is much smaller than the focal gene (possible indel)
`nst`	U_{ind} if search interval has an N-string
`scr`	U_{scr} if synteny is scrambled in the target context
`nm`	U_{nm} if the search interval is fine, but no match is found

[1] Statistically-significant sequence similarity (see **Section A.5**)

[2] Feature specified by GFF input for target species

[3] Sequence of codons bound by a START and STOP codon (no splice sites sought)

A: AAic - matches the translated product of an ORF

N: NTic - not AAic but matches a DNA region

U: unknown - no sequence similarity found in the search intervals

Figure 6.3 The default `fagin` decision tree for determining homology classes. The process first asks whether the focal gene has a significant aa match to an annotated protein in the `synder`-derived search interval of the target genome (green node, `gen`). If yes, the gene is classified as A_{gen}, otherwise, the next question is asked. This process continues along the decision tree until a homology class (red leaf) is assigned. `y`, yes; `n`, no. The tree can be modified or replaced by the user. For example, nodes with other evidence or analysis can be added, with associated homology classes.

found, in which case `fagin` will attempt to determine the biological or technical reason why no similarity was found (**Figure 6.2**). The assignments are made by following a binary decision tree (**Figure 6.3**). This tree may be customized. Here, we focus on the default tree of `fagin`.

6.2.3.1 AAic class

The amino acid sequence encoded by each query gene is searched against the translated CDSs and ORFs of the syntenic search intervals in each target species to infer the presence or absence of a potential ortholog (**Figure 6.2**). Following the decision tree, `fagin` divides the AAic class into three groups. A query gene is classified into the first affirmative case on the decision tree (see **Figure 6.3**).

The query gene is A_{gen} if the encoded protein of a query gene has amino acid similarity to an annotated protein of the target species that overlaps a syntenic search interval. This class is strong evidence that a query gene has an ortholog in the target. The next two classes, A_{trn} and A_{gen} are amino acid matches to *potential* coding sequences. A_{trn} indicates similarity to an ORF (other than the CDS) on an annotated mRNA, for example a short ORF in the 3' UTR. A_{orf} indicates similarity to a translated ORF that does not overlap an annotated mRNA. A_{orf} is an expected class for unannotated orthologs, rapidly-changing genes, and also potential *de-novo* orphans (comparison of similarity distribution, further RNAseq data, proteomic data, and experimentation could be used to test among these possibilities).

6.2.3.2 NTic class

If a query gene has no amino acid similarity to any CDS or ORF overlapping its target-side search interval, then evidence for nucleotide matches is sought. A focal gene is classified as N_{cds} if it contains a DNA match to a CDS that overlaps the target-side search interval (**Figure 6.2**, **Figure 6.3**). A query gene is classified as N_{exo} if it contains a DNA match to an exon that overlaps the target-side search interval. A query gene is

classified as *Nrna* if it contains a DNA match to an intron of any target gene that overlaps the search interval. Finally, a query gene is classified as N_{dna} if it contains a DNA match anywhere within its search interval that does *not* overlap any known gene; N_{dna} is an intergenic match.

Since NTic query genes have no amino acid similarity to any ORF in the search interval (a similarity would have led to an AAic classification), then the ortholog of the NTic focal gene is likely non-genic. NTic classifications are thus consistent with a *de novo* origin.

6.2.3.3 Unknown class

If the query gene has no significant amino acid or nucleotide similarity within its target-genome search interval (i.e., the query gene has an Unknown origin), then `fagin` will search for the most likely reason why no similarity was found. As with AAic and NTic classes, a query gene is classified into the first affirmative case on the decision tree (**Figure 6.3**).

Several cases are biologically interesting (**Figure 6.2**, even more so in comparison to the analogous results for conserved genes (see Results). The query gene is U_{ind} if its search interval on the target genome is much smaller than the query gene. This implies the ortholog may have been either deleted in the target genome or inserted in the focal genome (i.e., an indel). The query gene is U_{scr} if the order of elements in the chromosome near the focal gene is highly scrambled relative to the target genome. If the species provided to `fagin` are too distant for species to be conserved, then most genes will fall into this category; in near relatives, this might indicate a region of high chromosomal instability. The query gene is U_{nm} if it is in a syntenic region that is large enough to accommodate it, but no match is found. This could be due to a rapid mutational evolution such that the gene that can no longer be detected even with the

reduced search space and high resolution of `fagin`, or due to the gene having been translocated out of the region, perhaps with a transposon.

Several of the U classifications are due to technical aspects associated with the genome annotations or assemblies, or to the current inability of `fagin` to search very long search spaces. The query gene is U_{una} if it is inferred by `synder` to be in a search interval that is flush against an end of the scaffold of the target genome. This implies that the ortholog in the target genome may be missing from the target genome assembly. The query gene is U_{nst} if the search interval in the target genome contains a string of unknown bases (N characters). This is also a sign of an incomplete assembly. The query gene is U_{tec} if any search interval was skipped because it is too long for a `fagin` search. The current release of `fagin` relies on a Smith-Waterman alignment to determine similarity scores. The runtime of this algorithm increases with the product of the focal and target lengths. To avoid extremely long run-times, `fagin` has a cutoff for the largest space it will search. If many genes are classified into this category, then the user should increase the maximum search space threshold or modify `fagin` to use a faster algorithm. Membership in the U_{tec} category was almost non-existent for our two case studies.

6.2.4 Stage 3: Infer the origin of each query gene

In *Stage 3*, the assignments of each query gene from *Stage 2* is used to determine phylostrata for each gene and explore the level of support for assignments. In the default settings of `fagin`, a potential biological origin for each query feature is inferred by a "UNA" classification, based on the assignment of the query feature to Unknown, NTic, and/or AAic classes across lineages (**Figure 6.4**). The UNA classes collate information from across the tree into a single vector of labels representing level of support for the existence of a genic or non-genic homolog in each outgroup (i.e., one label for each node from focal species down to the root of the tree).

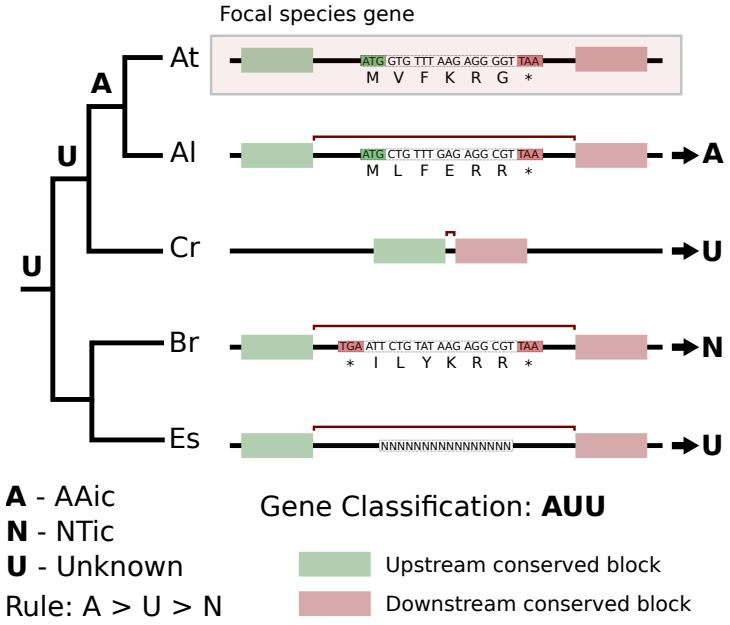


Figure 6.4 UNA classes. On the **left** is the Brassicaceae family tree. On the **right**, is the syntenic context for a imaginary query gene versus each target species. The query gene matches a gene in the most closely related target species, so is classified as AAic (A). In Cr, there is evidence of an indel, but no positive evidence for presence/absence of the gene in the species, so it is labeled as Unknown (U). The most distant branch from the focal species contains two species, one of which contains a positive NTic match and for other data is possibly missing. Since it is possible that an AAic match exists in Es, the branch is classified as Unknown overall. These three labels (A, N, and U) are a qualitative indication of the support for a gene being present along each cousin branch. The query gene is certainly not an orphan gene, but more precise statements are not justifiable. Including more target species could support a stronger inference for the query gene origin.

The query genes are assigned to UNA classes as follows. Let internal nodes from the focal species to the root of the species tree be numbered p_0 to p_K, where p_0 is the parent of the focal species, p_1 is the grandparent, and so on down the trunk of the tree to the root, p_K. Borrowing genealogical terminology, a set of "cousin" species can be defined for each ancestor (p_i) of the focal species. The 0th cousins, t_0, (i.e., siblings) are the species descending directly from p_0. The 1st cousins, t_1, are the species descending from p_1, and so on to the Kth cousins. The goal is to determine which ancestor, p_i, first possessed the gene in a coding state; that is, find i where the ancestral species p_i has a protein-coding homolog of the query gene but where p_j does not for all $i < j \leq K$.

For each ancestor, `fagin` infers whether an orthologous coding gene could have existed. To this end, we collapse the homology class of each species tree, from t_0 to t_K, to a single homology class. If we assume that the event leading to the origin of the ancestor of the focal species gene occurred only once (i.e., a single-birth model), then AAic classes should appear only in cousins descending from the ancestor that had the original gene. Under this assumption, if *any* leaf in the t_i tree is classified as AAic, then the entire subtree is classified as AAic. If *all* leaves in the subtree are NTic, then the subtree is classified as NTic. In cases where the leaves include at least one unknown and zero or more NTic, the entire tree is classified as unknown, since the unknown gene *could* be AAic. This is a stringent rule that is biased to a high estimation of uncertainty.

In summary, the subtree classification rule is:

$$s_{ij} = \begin{cases} A & \text{if any leaf in } t_i \text{ is AAic relative to the } j\text{th focal feature} \\ N & \text{if all leafs in } t_i \text{ are NTic relative to the } j\text{th focal feature} \\ U & \text{otherwise} \end{cases} \tag{6.1}$$

Where s_{ij} is the label assigned to the ith cousin subtree (or the ith position in the UNA vector).

Following this pattern, a UNA classification, a vector of length $K+1$, can be inferred for each subtree (see **Figure 6.4**). The gene can be classified into a synteny-based phylostratum for gene j by finding the maximum i such that $s_{ij} = A$ and $s_{zj} \neq A \; \forall i < z \leq K$. For example, if there is support across all nodes for the AAic class, from siblings to most distant cousins, we can infer that the earliest common ancestor was genic.

Alternate ways to infer the origin of gene features based on multiple target genomes are possible, and can be customized in `fagin`. For example, the classification could take into account the length of the matches to the target genome ORF, or it could incorporate the sub-classifications of AAic, NTic and Unknown.

This approach to inferring gene origin can be considered a significant modification of standard phylostratigraphy. In standard phylostratigraphy, the proteomes of related species are searched for similarity to a focal gene. If a significant hit is found, the species is classified as having a homolog. This classification is similar to the `fagin` AAic classification, *except* that in `fagin`: 1) the search is restricted to *syntenically* matching regions; and; 2) the amino acid hits may correspond to annotated CDS, unannotated ORFs on known mRNAs, or unnannotated mono-exonic ORFs anywhere in the search interval; and 3) a distinction is made between classifications based on positive evidence (i.e., A or N) and those based on negative evidence (U).

Thus, whereas standard phylostratigraphy is based on a binary decision about the presence or absence of a homolog [32], and synteny-based *de* novo gene pipelines classify the *matches* in the syntenic search interval (e.g., [37, 39]), `fagin` is based on a three way decision, followed by subclassifications: 1) a possible protein-coding match; 2) positive evidence that there is *no* protein-coding match; and 3) no answer can be found. Essentially, standard approaches merges the `fagin` categories N and U, and thus does not distinguish between matches that are missed due to bad data and matches that are missed due to absence of the gene.

6.3 Results

We demonstrate use of the `fagin` pipeline on two focal species: *S. cerevisiae* and *A. thaliana*. These species have been analyzed using standard phylostratigraphy in earlier papers identifying 423 Saccharomyces-specific genes [43] and 2425 Brassicaceae-specific genes [44]. Building off these prior studies, clade-specific genes were fed into the `fagin` pipeline for deeper analysis. Both focal species have good genome assemblies, but the target species in each study were of variable quality (see **Table 6.1**). We built pairwise synteny maps between the focal genomes and each target using MUMmer4 [45] (for Saccharomyces) and Satsuma [42] (for Brassicaceae). The synteny maps where fairly dense, with several hundred blocks per megabase and block length medians ranging from 102 to 389 (see **Section A.1**).

`fagin` first infers homology classes between two species. From the homology classes, we infer the phylostrata for each focal gene and compare them to those inferred through standard methods. Finally, we break the phylostrata into finer classes based on UNA vectors.

6.3.1 A summary of homology classes

The homology classes for the Saccharomyces and Brassicaceae studies are summarized in **Figure 6.5**. Summaries of the search interval lengths and inferences about syntenic ambiguity or genome assembly issues is available in **Section A.2**. In each study, all orphan genes, all lineage-specific genes (unique to genus for Saccharomyces, unique to family for Brassicaceae), and a random sample of ancient genes, as inferred by standard phylostratigraphy [43], were passed through the `fagin` pipeline. As expected, the majority of the ancient genes fall into the AAic class (see the **ancient** rows of bar plots in **Figure 6.5**). However, about 20% of ancient *S. cerevisiae* query genes are classified as A_{orf} relative to *S. arboricola*. This implies a strong disagreement between the gene an-

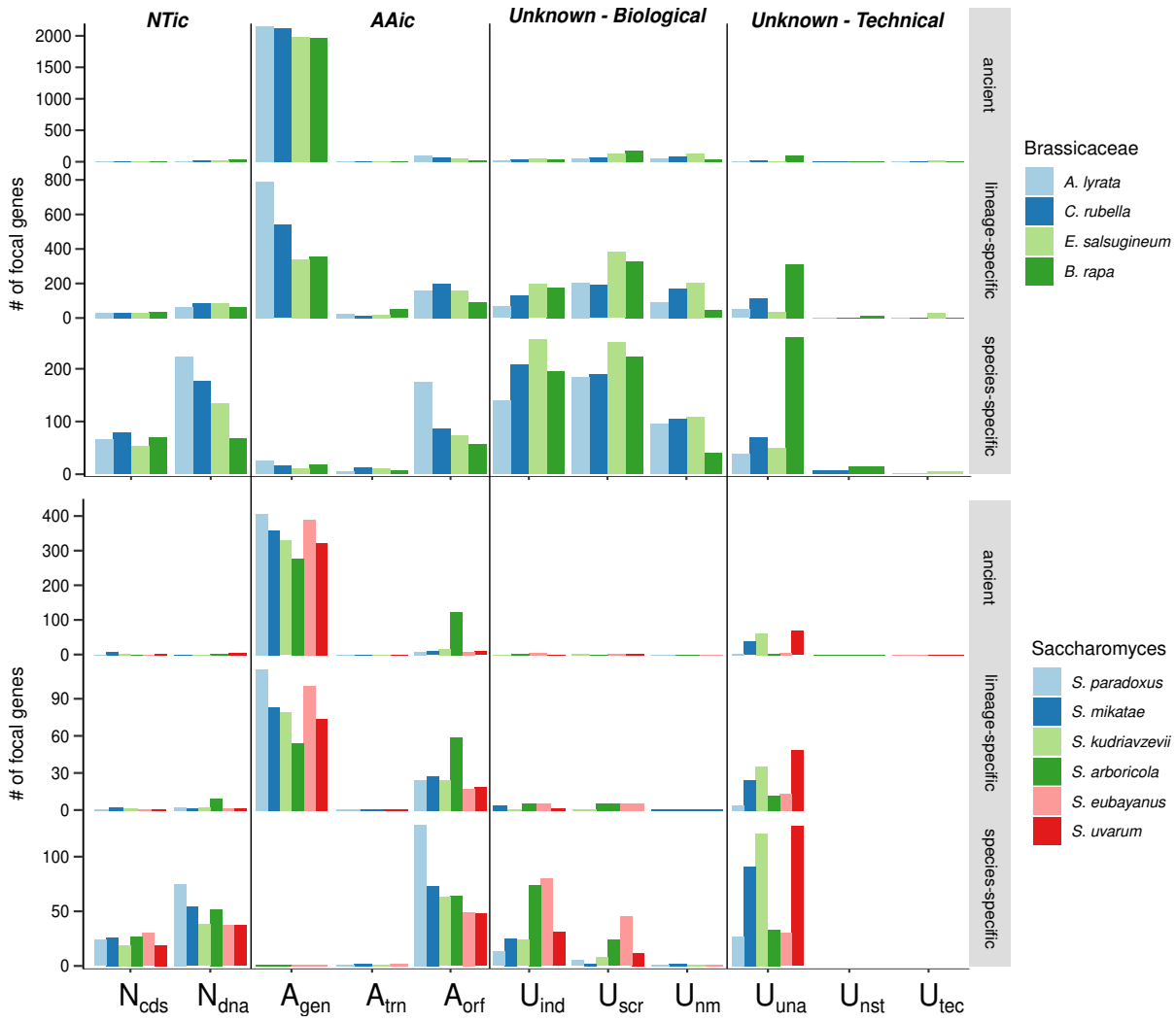


Figure 6.5 `fagin`-inferred homology classes for the Brassicaceae (**above**) and Saccharomyces (**below**) case studies. In each study, the top row of plots, labeled **ancient**, represents a random sample of genes from ancient strata (outside Brassicaceae or Saccharomyces). The **lineage-specific** row includes genes that are unique to the clade, but that are not unique to the focal species. The **species-specific** row includes only the orphan genes. The original inferences of orphan, lineage-specific, and ancient genes were made by standard phylostratography [43]. Each group of bars represents the number of query genes that fall into a given homology class in relation to each target species

Table 6.1 Genomic statistics for species in the Brassicaceae (top) and Saccharomyces (bottom) case studies. **nseq**, number of scaffolds in the assembly; **n50**, number of bases in the scaffold that contains the genomic midpoint in a list of scaffolds sorted by length; **size**, size of the genome; **prots**, number of gene models in the genome; **Ns**, number of unknown bases (N) in the genome assembly.

species	nseq	n50 (nt)	size (nt)	prots	Ns
A. thaliana	7	23459830	119667750	35386	185738
A. lyrata	695	24464547	206667935	32550	22960134
C. rubella	773	15040190	133063876	28713	3314705
E. salsugineum	638	13441892	243110105	29485	4665582
B. rapa	40249	26286742	284129391	51005	10904295
S. cerevisiae	17	924431	12.2M	6008	0
S. paradoxus	832	49124	11.9M	5933	0
S. mikatae	1648	20026	11.5M	6086	0
S. kudriavzevii	2054	11253	11.2M	6529	2127
S. arboricola	35	879294	11.6M	3659	224325
S. eubayanus	24	896107	11.7M	5379	121986
S. uvarum	1098	25082	11.5M	5721	0

notations in the focal species, *S. cerevisiae*, and the *S. arboricola* target species; indeed, only 3,659 genes are annotated in *S. arboricola* (**Table 6.1**).

In both case studies, a high proportion of the orphan genes are classified into the Unknown category, predominantly U_{ind} (indels), U_{scr} (syntenically scrambled), U_{una} (bad assembly), and U_{nm} (no match found). These subclassifications can be informative. *B. rapa*, the target species with the most incomplete assembly (see **Table 6.1**), also has the highest number of orphan genes missing due to bad assembly. A more interesting case is that many orphan genes from both *S. cerevisiae* and *A. thaliana* fall into Unknown-biological categories; this indicates the intriguing possibility that orphans may be associated with regions of genome change.

6.3.2 Synteny-based phylostratigraphy

We compare standard phylostratigraphy results to two fagin-based approaches (**Figure 6.6**). The first fagin-based approach, fagin-default, infers homologs in target species

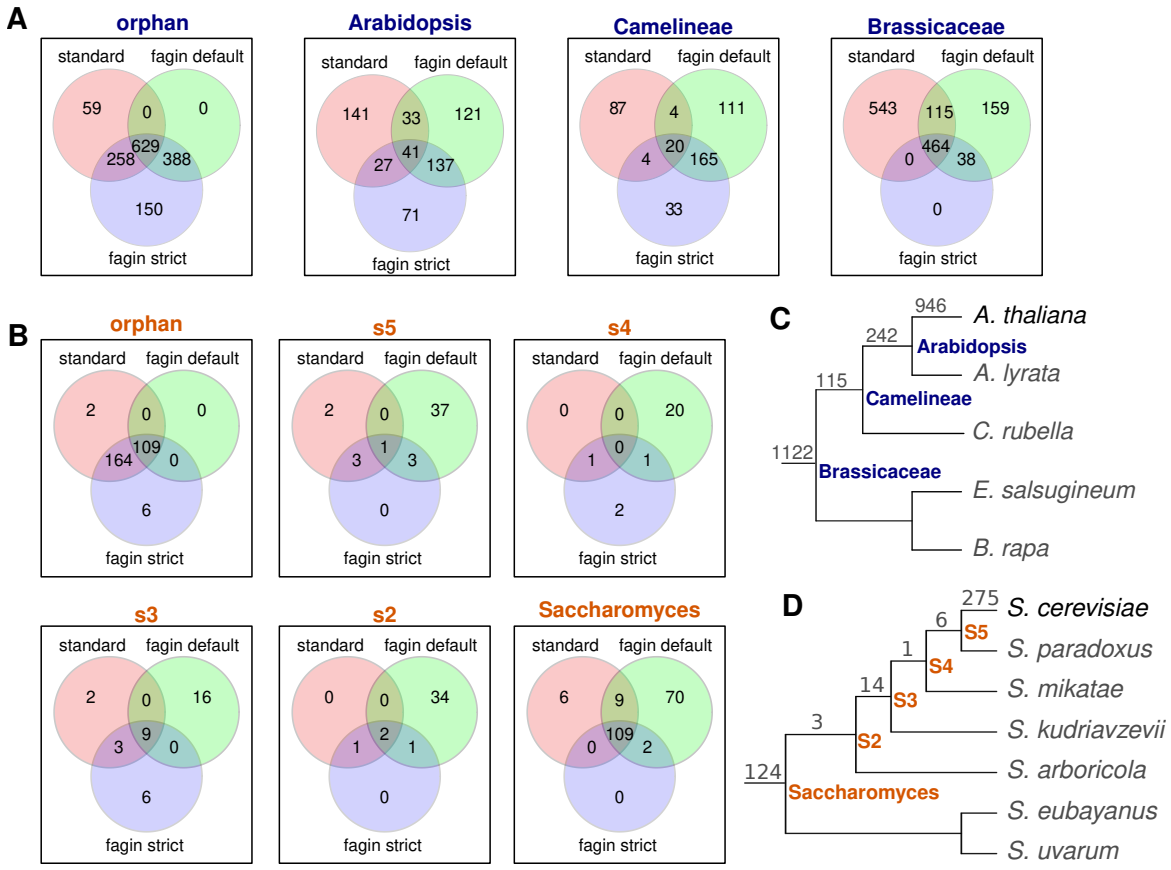


Figure 6.6 Comparison of assignments in gene classifications by three methods. The Brassicaceae study (**A**) represents overlaps in gene classifications across four phylostrata of Brassicaceae. The Saccharomyces study (**B**) represents overlaps in gene classifications across six phylostrata, from the *S. cerevisiae*-specific **orphan** phylostrata, through the genes unique to each of the **s5** to **s2** internal clades, to the genes conserved across the *Saccharomyces* genus. The three methods of comparison are 1) **standard** which represents standard phylostratigraphy; 2) **fagin default** which is the default `fagin` behaviour of identifying phylostrata based on presence/absence of any AAic inferred ortholog; and 3) **fagin strict** which identifies phylostrata based on presence/absence of amino acid matches only to annotated target genes (similar to standard phylostratigraphy). All methods use the set of protein coding genes that were inferred through standard phylostratigraphy to be limited to the Brassicaceae or Saccharomyces clades. **C** and **D** are the species trees representing the target genomes used for Brassicaceae [46] and Saccharomyces [47], respectively. The numbers indicate the number of genes in each clade according to standard phylostratigraphy (from `phylostratr` for Saccharomyces [43]; [44] for Brassicaceae). Nodes on the Saccharomyces tree, orange text, are labeled as S5-S2 because there are no names for these within-genus clades.

based off all three AAic classes. The second fagin-based approach, fagin-strict, infers homologs based only on matches of the query amino acid sequence to known protein-coding genes in the target species (the A_{gen} class); this roughly emulates standard phylostratigraphy but is limited to syntenic genes.

In the Saccharomyces study (**Figure 6.6B**), the fagin-strict classifications agree closely with standard phylostratigraphy. However, fagin-default infers older origins for 164 genes that were classified as orphans by standard and fagin-strict. The majority of these genes were inferred as being older by fagin-default due to an amino acid match of the query gene to the amino acid predicted sequence of an unannotated non-genic ORF (A_{orf}). There are two interesting interpretations. 1) An orthologous gene might be located in the syntenic region of the target species but it might not be annotated as a gene. In this case, the gene would not be an orphan, but rather would be older. Standard phylostratigraphy would not detect these homologs since only annotated genes are searched. 2) The match might be to the predicted amino acid sequence of an untranscribed and/or untranslated ORF. In this case, the query gene might be a very-rapidly-evolving orphan, i.e., an orphan that did not originate *de novo* during or post-speciation. It is difficult to detect genes that are rapidly changing, and the mechanisms for this change are also interesting [48]. Possible methods to gain insight into which target-side ORFs are real hits would be to compare the size of the ORF to the ORF of the focal gene, to assess evidence of transcription and translation, and to look for evidence of selection.

Among the *S. cerevisiae* orphan genes, there is one gene that is unique to standard analysis and six that are unique to fagin-strict. The genes uniquely designated as orphans by fagin-strict could be genes that hopped out of context (e.g., transposed) and were thus absent from the syntenic search space. A search for transposon footprints might reveal if this was the case. The gene uniquely designated by standard phylostratigraphy as an orphans is possibly a case where the reduced search space, and resulting higher

statistical resolution, led to an inferred homology that could not be detected in the standard phylostratigraphy search against the full target proteome.

In the Brassicaceae study, the standard and fagin-strict methods obtained very different results. Since the main difference between the two methods is that fagin-strict is limited to searching syntenic genes, the search intervals inferred from the synteny maps must be missing many of the true orthologs. `fagin`-based phylostratigraphy treats genes that cannot be found as genes with no homologs. Many of the query genes have Unknown homology-classes across all target species and are thus classified as orphans. Thus the genes of unknown origin and the genes of confirmed recent origin are pooled. To resolve these groups, we can look deeper into the gene classifications `fagin` provides.

6.3.3 Finer grain analysis of phylostrata with UNA classes

The homology classes contain much information that is lost when reducing down to just phylostratigraphy labels. We can gain more insight into the support for the phylostrata classes by looking at the *UNA* vectors (see **Section 6.2.4**). A summary of UNA classes for the Brassicaceae study is shown in **Table 6.2**. This table partitions all the genes in the focal species into the four phylostrata as well as a fifth class where there is no evidence for a syntenic homolog even in the closest relative.

Among the Brassicaceae-specific genes as inferred by `fagin` (**Table 6.2**, Brassicaceae-specific), there are 474 genes classified as AAA. These are genes with strong positive evidence of being present across the Brassicaceae clade. In contrast, the 34 NNA are possibly orphan genes, in which the deepest A is likely a false positive, such as a match to a non-genic ORF that is actually non-functional. The 26 UUA genes are in between, with weak evidence for their Brassicaceae-spanning classification. Incorporating additional genomes into the analysis might help resolve these disparities.

The 48 NNN query genes are the most strongly supported *de novo* orphan genes. The 162 NUU, 80 NNU, and 4 NUN genes are also supported *de novo* orphans, for which

Table 6.2 UNA labels for Brassicaceae ordered by phylostratum. The **Brassicaceae-specific** column contains counts of query genes with each UNA label from among genes that is inferred by standard phylostratigraphy to be Brassicaceae-specific. The **non-Brassicaceae-specific** column contains counts of older genes that are used as a control. The **phylostratum** column contains the phylostratum as inferred by the deepest character in the UNA vector that is AAic (the **A** in bold).

Class	Brassicaceae-specific	non-Brassicaceae-specific	phylostratum
AA**A**	474	2033	Brassicaceae
AN**A**	52	7	
UA**A**	26	18	
AU**A**	44	44	
NA**A**	18	4	
NN**A**	34	0	
NU**A**	3	3	
UN**A**	8	0	
UU**A**	26	14	
A**A**U	194	111	Camilineae
A**A**N	20	3	
N**A**N	7	0	
N**A**U	27	3	
U**A**U	27	17	
ANN	38	0	Arabidopsis
ANU	47	6	
AUN	5	0	
AUU	256	58	
NNN	48	1	*A. thaliana*
NNU	80	0	
NUN	4	0	
NUU	162	4	
UNU	25	0	Unknown
UUU	800	99	

analysis of more target genomes could provide more support. A particularly interesting class of genes are the lineage-specific genes of *de novo* origin with the labels ANN and AAN. These are genes with positive evidence of being *de novo*, having evolved from non-genic precursors and survived to spread across several species. These *de novo* genes could be studied to shed light on the dynamics and evolution of the functional evolution of *de novo* genes.

The 800 UUU genes are genes with no positive evidence of being present in any form outside *A. thaliana*. Standard phylostratigraphy did not detect them in any species, and they have no syntenic homologs. All these are candidate orphans. `fagin` can offer hints about the origin of these UUU genes, from their synteny-based U sub-classes. A deeper look into the sub-classes, and further analysis of the search intervals, could give us a better understanding of the origin of each of these genes. Some may be missing for technical reasons (incomplete assemblies) while others may be missing for more interesting biological reasons (rapid syntenic rearrangements or transposition).

The UNA classes can of course be further broken down on a gene-by-gene basis into the homology classes. The actual alignments from all the homology searches is stored by `fagin`. All of this data can serve as a starting point for deeper analysis of the origins of specific genes.

6.4 Discussion

A key difference between synteny-based phylostratigraphy and standard phylostratigraphy is the emphasis on positive evidence [37, 39]. The methods differ in two significant ways. First, the synteny-based approach is more sensitive, since it searches the small, synteny-based search space, instead of the entire proteome. This effectively leads to younger classifications. Second, by limiting the search to syntenic regions, it both avoids false positives and, when synteny is unclear, misses true positives — in either case, the synteny-based approach infers younger classes. However, analysis of gene age based

on synteny has the limitation that it is restricted to only those cases when synteny is reasonably conserved. For example, synteny may be sufficiently conserved across the primate family, but probably not across Animalia. In this sense, syntenic analysis uncovers only recent evolutionary events. Similarly, a target genome may have undergone multiple rearrangements, and for some regions of that genome no syntenic region might be identifiable. Thus, each method has strengths and weaknesses. We suggest that using genome-wide analysis along with more in-depth syntenic analysis can address the individual limitations.

Those genes designated by standard phylostratigraphy as "orphans" but that have no observed synteny to a region in a sister genome are interpreted differently by different researchers. In some studies, they are relegated explicity or implicitly, to the large group of "genes of unknown origin". Other studies classify all genes with no detectable homology in related species as orphans (i.e., studies in which the classification "orphan" depends solely on absence of significant similarity to an annotated protein). Still other studies have categorized the genes with no syntenic matches as not being *de novo* orphan genes (e.g., [38]). In actuality, if the gene cannot be traced, the origin is unclear. With `fagin`, although no origin is assigned, genes without syntenic matches in a target species are classified as missing for biological reasons (such as deletion events or evolution beyond recognition) or technical reasons (such as missing sequence or poor assembly). Thus, the `fagin` subcategories resolve genes with no syntenic match into specific inferred phenomena, such as deletion/insertion events, missing sequence or poor assembly.

Synteny has been an important part of research to identify genes of *de novo* origin [37, 7, 25, 38]. `fagin` makes the technique automatic, general, and reproducible. Further, it extends the technique, by offering a deeper analysis of the source and magnitude of the classification error. `fagin` can be applied to annotate orphans of *de novo* origin in new genome sequencing projects, to identify promising orphan gene candidates for further experimental research, and to directly study the dynamics of *de novo* gene evo-

lution. Likewise, it can provide candidates for proteins that are targets of ultra-rapid evolution-beyond-recognition. In particular, those genes that are classified by standard phylostratigraphy as orphans, but show an amino acid match to the CDS of a known gene in the more sensitive search to a syntenic interval of a target genome are candidates for being rapidly changing genes.

`fagin` differs from current syntenic approaches in several ways. First, it enables a user to seamlessly go from data input to final results and summaries. Second, the `fagin` pipeline is flexible and easily modified. A user can compare various methods for classification of the same genome data sets, or evaluate classifications based on different determinations of the syntenic search space used for each query gene. A user also can choose to extract intermediate data from any step in the pipeline. Third, `fagin` classifies every gene by probing the syntenic space of each query gene and explicitly distinguishing among query genes that have an amino acid match, those that have only a nucleotide match, and those that have no match, i.e., are of unknown origin. These classes are then sub-categorized, inferring extensive information about each gene's origin. Finally, `fagin` uses multiple target genomes, thus providing additional evidence to support query gene classifications. These later features help to highlight the ambiguity of assignments, and the challenges of working with complex biology and incomplete data.

`fagin` can also be used to study overprinting, the phenomenon in which a single gene encodes more than one protein or one reading frame gives way to another over evolutionary time. Overprinting is a common scenario in viruses, in which many such overprinted proteins are orphans [49, 50]. Though less studied, overprinting also occurs in Eukaryotes [51] and may be involved in *de novo* gene origin [6]. The signature of an overprinted gene in `fagin` would be a query gene that does not match any annotated (target-side) syntenic coding gene but that does match a transcribed ORF that overlaps a known gene (i.e., N_{cds} class).

`fagin`'s consideration of multiple genomes facilitates comparisons of orthologs across evolutionary time. Specifically, `fagin` will allow for the systematic identification and study of lineage-specific genes of *de novo* origin that are conserved across a subclade of the family, but non-genic outside the subclade. (Subclades also can include within-species populations.) Since lineage-specific *de novo* genes have homologs, they can be studied in their evolutionary context. This is in contrast to species-specific genes, where even the gene's status as a *bona fide* gene is often difficult to prove. Analysis of the sequence of these older *de novo* genes in different lineages will shed light on how they evolved. Do they specialize their expression patterns and functions in different lineages? How does their disappearance/deletion rate compare to that of older genes? Do they become longer and more complex over time? Do their codons become more optimized? How do the properties of these genes change is relation to those of rapidly evolving genes of more ancient origin? By automating the complex process of syntenic phylostratigraphy, `fagin` will allow such studies to be done on a large scale. This will be a four step process: 1) collect data for all members of each focal and target genome in a clade; 2) construct pairwise synteny maps between focal and target genomes; 3) run a standard phylostratigraphy study (this may be automated with `phylostratr` [43]); and 4) run the family-specific genes through `fagin`.

`fagin`'s generalizable structure greatly simplifies additions and extensions. In particular, the flexible decision tree is foundational to `fagin`. The decision tree for determining homology classes can be altered by adding additional nodes that contain different data-types or rules. The structure of this tree is central to simplifying writing extensions and making changes. `fagin` could be merged with `phylostratr` to integrate synteny-based phylostratigraphy for shallow clades with standard phylostatigraphy for deeper clades. The tree could be adjusted to follow the analysis pipeline suggested in [39]. Or, transcriptomics data could be interpreted to indicate which ORFs are transcribed, and this information could be added to `fagin`. Support could be added to incorporate evidence

of translation, such as ribosome footprinting or proteomic mass spectroscopy data. Phenotype evidence could be added. The analysis of ORFs could be extended to predictions of unannotated, spliced, transcripts. Adding new nodes to the decision tree would also add new classes of orthologs in the target genome, with richer information and support. For example, adding a new node for transcriptomic data and one for proteomic data would provides two new AAic classes of orthologs: one for unannotated ORFs with experimental evidence of transcription, and one for unannotated ORFs with proteomic support.

We anticipate `fagin` will serve as a general framework for phylostratigraphy and orthology inference, providing a consistent and reproducible way to compare mechanisms of evolutionary change across genomes. Since `fagin` relies on synteny, it will become increasingly useful as the number and quality of genome sequences rises.

Author Contributions

ZA and EW conceived of the project and ZA implemented the software. US, JL, PB, and AS tested the package and helped with case studies. ZA and EW wrote the manuscript and all authors read and commented on it.

Bibliography

[1] Jacob, F. (1977) Evolution and tinkering. *Science* 196, 1161–1166

[2] Fischer, D. and Eisenberg, D. (1999) Finding families for genomic ORFans. *Bioinformatics (Oxford, England)* 15, 759–762

[3] Chen, L. *et al.* (1997) Evolution of antifreeze glycoprotein gene from a trypsinogen gene in Antarctic notothenioid fish. *Proceedings of the National Academy of Sciences* 94, 3811–3816

[4] Ruiz-Orera, J. *et al.* (2015) Origins of de novo genes in human and chimpanzee. *arXiv preprint arXiv:1507.07744*

[5] Zhao, L. *et al.* (2014) Origin and spread of de novo genes in *Drosophila melanogaster* populations. *Science* 343, 769–772

[6] Neme, R. and Tautz, D. (2013) Phylogenetic patterns of emergence of new genes support a model of frequent de novo evolution. *BMC genomics* 14, 117

[7] Donoghue, M.T. *et al.* (2011) Evolutionary origins of brassicaceae specific genes in *Arabidopsis thaliana*. *BMC evolutionary biology* 11, 47

[8] Yang, Z. and Huang, J. (2011) De novo origin of new genes with introns in *Plasmodium vivax*. *FEBS Letters* 585, 641–644

[9] Carvunis, A.R. *et al.* (2012) Proto-genes and de novo gene birth. *Nature* 487, 370–374

[10] Tautz, D. and Domazet-Lošo, T. (2011) The evolutionary origin of orphan genes. *Nature Reviews Genetics* 12, 692–702

[11] Bhandary, P. *et al.* (2017) Raising orphans from a metadata morass: a researcher's guide to re-use of public 'omics data. *Plant Science*

[12] Qi, M. *et al.* (2018) QQS orphan gene and its interactor NF-YC 4 reduce susceptibility to pathogens and pests. *Plant biotechnology journal*

[13] Voolstra, C.R. *et al.* (2011) Rapid evolution of coral proteins responsible for interaction with the environment. *PLoS ONE* 6, e20392

[14] Colbourne, J.K. *et al.* (2011) The ecoresponsive genome of *Daphnia* pulex. *Science* 331, 555–561

[15] Li, L. *et al.* (2009) Identification of the novel protein QQS as a component of the starch metabolic network in Arabidopsis leaves. *The Plant Journal* 58, 485–498

[16] Khalturin, K. *et al.* (2009) More than just orphans: are taxonomically-restricted genes important in evolution? *Trends in Genetics* 25, 404–413

[17] Johnson, B.R. and Tsutsui, N.D. (2011) Taxonomically restricted genes are associated with the evolution of sociality in the honey bee. *BMC genomics* 12, 164

[18] Andrikou, C. and Arnone, M.I. (2015) Too many ways to make a muscle: Evolution of GRNs governing myogenesis. *Zoologischer Anzeiger - A Journal of Comparative Zoology* 256, 2–13

[19] Tomalova, I. *et al.* (2012) The map-1 gene family in root-knot nematodes, Meloidogyne spp.: a set of taxonomically restricted genes specific to clonal species. *PLoS ONE* 7, e38656

[20] Wheeler, M.J. *et al.* (2009) Identification of the pollen self-incompatibility determinant in *Papaver rhoeas*. *Nature* 459, 992

[21] Xiao, W. *et al.* (2009) A rice gene of de novo origin negatively regulates pathogen-induced defense response. *PLoS ONE* 4, e4603

[22] Kohler, A. *et al.* (2015) Convergent losses of decay mechanisms and rapid turnover of symbiosis genes in mycorrhizal mutualists. *Nature Genetics*

[23] Shigenobu, S. and Stern, D.L. (2012) Aphids evolved novel secreted proteins for symbiosis with bacterial endosymbiont. *Proceedings of the Royal Society B: Biological Sciences* 280, 20121952–20121952

[24] Li, L. and Wurtele, E.S. (2015) The QQS orphan gene of Arabidopsis modulates carbon and nitrogen allocation in soybean. *Plant biotechnology journal* 13, 177–187

[25] Xie, C. *et al.* (2019) Studying the dawn of de novo gene emergence in mice reveals fast integration of new genes into functional networks. *bioRxiv* p. 510214

[26] Oldenburg, K.R. *et al.* (1992) Peptide ligands for a sugar-binding protein isolated from a random peptide library. *Proceedings of the National Academy of Sciences* 89, 5393–5397

[27] Keefe, A.D. and Szostak, J.W. (2001) Functional proteins from a random-sequence library. *Nature* 410, 715

[28] Whaley, S.R. *et al.* (2000) Selection of peptides with semiconductor binding specificity for directed nanocrystal assembly. *Nature* 405, 665

[29] Surdo, P.L. *et al.* (2004) A novel adp-and zinc-binding fold from function-directed in vitro evolution. *Nature Structural and Molecular Biology* 11, 382

[30] Neme, R. *et al.* (2017) Random sequences are an abundant source of bioactive RNAs or peptides. *Nature ecology & evolution* 1, 0127

[31] Bao, Z. *et al.* (2017) Identification of novel growth regulators in plant populations expressing random peptides. *Plant physiology* pp. pp–00577

[32] Domazet-Lošo, T. *et al.* (2007) A phylostratigraphy approach to uncover the genomic history of major adaptations in metazoan lineages. *Trends in Genetics* 23, 533–539

[33] Šestak, M.S. and Domazet-Lošo, T. (2015) Phylostratigraphic profiles in Zebrafish uncover Chordate origins of the Vertebrate brain. *Molecular Biology and Evolution* 32, 299–312

[34] Domazet-Loso, T. and Tautz, D. (2010) Phylostratigraphic tracking of cancer genes suggests a link to the emergence of multicellularity in metazoa. *BMC biology* 8, 66

[35] Jain, A. *et al.* (2018) The evolutionary traceability of proteins. *bioRxiv*

[36] Moyers, B.A. and Zhang, J. (2014) Phylostratigraphic bias creates spurious patterns of genome evolution. *Molecular biology and evolution* 32, 258–267

[37] Knowles, D.G. and McLysaght, A. (2009) Recent de novo origin of human protein-coding genes. *Genome Research* 19, 1752–1759

[38] Casola, C. (2018) From de novo to "de nono": The majority of novel protein-coding genes identified with phylostratigraphy are old genes or recent duplicates. *Genome biology and evolution* 10, 2906–2918

[39] Vakirlis, N. and McLysaght, A. (2019) *Computational Prediction of De Novo Emerged Protein-Coding Genes*, pp. 63–81. Springer New York, New York, NY

[40] Arendsee, Z. *et al.* (2019) synder: inferring genomic orthologs from synteny maps. *bioRxiv*

[41] Marçais, G. *et al.* (2018) MUMmer4: A fast and versatile genome alignment system. *PLoS computational biology* 14, e1005944

[42] Grabherr, M.G. *et al.* (2010) Genome-wide synteny through highly sensitive sequence alignment: Satsuma. *Bioinformatics* 26, 1145–1151

[43] Arendsee, Z. *et al.* (2019) phylostratr: A framework for phylostratigraphy. *Bioinformatics*

[44] Arendsee, Z.W. *et al.* (2014) Coming of age: orphan genes in plants. *Trends in plant science* 19, 698–708

[45] Marçais, G. *et al.* (2018) MUMmer4: A fast and versatile genome alignment system. *PLoS computational biology* 14, e1005944

[46] Yang, R. *et al.* (2013) The reference genome of the halophytic plant *Eutrema salsugineum*. *Frontiers in Plant Science* 4, 46

[47] Shen, X.X. *et al.* (2016) Reconstructing the backbone of the Saccharomycotina yeast phylogeny using genome-scale data. *G3: Genes, Genomes, Genetics* pp. g3–116

[48] Zhuang, X. *et al.* (2019) Molecular mechanism and history of non-sense to sense evolution of antifreeze glycoprotein gene in northern gadids. *Proceedings of the National Academy of Sciences* 116, 4400–4405

[49] Rancurel, C. *et al.* (2009) Overlapping genes produce proteins with unusual sequence properties and offer insight into de novo protein creation. *Journal of virology* 83, 10719–10736

[50] Sabath, N. *et al.* (2012) Evolution of viral proteins originated de novo by overprinting. *Molecular biology and evolution* 29, 3767–3780

[51] Chung, W.Y. *et al.* (2007) A first look at ARFome: dual-coding genes in mammalian genomes. *PLoS computational biology* 3, e91

APPENDIX A. `fagin`: SUPPLEMENTARY MATERIAL

Supplementary Material

A.1 Synteny map construction and summaries

Table A.1 Numeric summary of the lengths of the syntenic blocks (in nucleotides) in the synteny maps between the focal species (*A. thaliana* and *S. cerevisiae*) and each of the listed target species for the Brassicaceae (**top**) and Saccharomyces (**bottom**) case studies. **N**, total number of blocks in the synteny map.

	Synteny Blocks (nt)					
Species	min	q25	median	q75	max	N
A. lyrata	25	33	163	365	9280	229562
C. rubella	25	39	197	399	6264	151688
L. rapa	25	150	214	366	10260	195085
E. salsugineum	25	146	222	400	8248	131874
S. paradoxus	63	177	389	852.00	13716	12232
S. mikatae	63	80	275	552.25	6836	6688
S. kudriavzevii	65	75	257	489.00	4870	5189
S. arboricola	65	71	159	417.00	4103	4534
S. eubayanus	65	70	102	375.00	4751	3914
S. uvarum	65	70	104	376.00	6836	3833

A.2 `synder` results and summary

`fagin` infers the search intervals by calling the `synder` search function. This is a function of a synteny map and four parameters: 1) `trans` specifies the function needed to transform the score column in the synteny map to one that is additive; 2) k is the number of conflicting syntenic intervals allowed in a block before it is broken (set to 0 by default); 3) r is a score decay rate that is used in calculating a score for each syntenic block created by `synder`; 4) `offsets` which specify the input and output bases (0 or 1)

Table A.2 Numeric summary of the lengths of the search intervals inferred by `synder` [1] between the focal species (*A. thaliana* and *S. cerevisiae*) and each of the listed target species for the Brassicaceae (top) and Saccharomyces (bottom) case studies. Column 2-7 refer to the minimum, 25th quantile, median, 75th quantile, and maximum of the block lengths. The final column, **N**, is the total number of blocks in the synteny map.

	Search Intervals (nt)					
Species	min	q25	median	q75	max	N
A. lyrata	1	807	1654	2949	2909847	72713
C. rubella	2	1040	2004	3440	216346	64133
S. rapa	1	826	1737	4057	298748	148594
E. salsugineum	2	1150	2299	4617	900973	66102
S. paradoxus	1	670.0	1215	2211	26047	13219
S. mikatae	1	556.5	1714	3858	31027	16039
S. kudriavzevii	1	1006.0	2332	4919	26409	21460
S. arboricola	1	1956.0	4643	9723	51035	19660
S. eubayanus	3	2480.0	5967	12521	61711	23995
S. uvarum	1	1327.5	3419	7746	45931	26095

of the synteny map. Parameters 1 and 4 are specific to the tool that created the synteny map.

The output of the `synder search` function is: 1) a set of one or more search intervals for each focal gene (summarized in **Table A.2**) and 2) a description of each search interval consisting of a flag describing each edge of the interval, whether the search interval overlaps a syntenic region in the synteny map, and a relative score for the search interval. The search intervals can be classified as shown in **Table A.3**.

A.3 `fagin` validation and parsing of GFF files

The GFF format is simple but highly error prone and GFFs from difference sources can follow very different conventions. This complicates practical analysis. While more standardized formats exist for storing feature information, GFF has persisted as the most commonly used. For this reason, `fagin` uses GFF, but also performs extensive validation and cleaning. The most problematic component of GFF is the 9th column that stores tag-value data. It is from these tag-value pairs that we build the gene models that we

Table A.3 Summary of `synder` flags [1] for the Brassicaceae (**top**) and Saccharomyces (**bottom**) case studies. **Between** means that the query gene was between but did not overlap any synteny block. **Lo**, **Hi**, **Both**, and **None** relate to whether the edges of the search interval are in a syntenically unambiguous region. A gene is counted as **Lo** if any search interval lower bound is unambiguous, **Hi** is any upper bound is unambiguous, **Both** if both bounds of its syntenic search interval is unambiguous, and **None** if no edge in its syntenic search interval(s) is unambiguous. **Scrambled** means all syntenic search intervals for that gene have ambiguous edges and are between syntenic links (i.e. are Between). **Unassembled** means the gene may be in an unassembled region of the genome (one edge of the search interval is flush against a terminus of a scaffold).

	`synder` search interval classifications						
Species	Between	Lo	Hi	Both	None	Scrambled	Unassembled
A. lyrata	3048	32701	32789	17514	1550	1749	871
C. rubella	5139	32140	32021	18747	2116	2573	754
B. rapa	7404	22241	22331	6778	7810	6404	4788
E. salsugineum	7545	30161	30227	17304	3586	3976	306
S. paradoxus	699	5822	5814	5226	340	328	1038
S. mikatae	3266	3980	4003	3370	2003	2027	3027
S. kudriavzevii	4135	2628	2655	2022	3312	3411	4568
S. arboricola	4488	5137	5154	4856	1101	2149	513
S. eubayanus	4972	4729	4724	4462	1541	2883	481
S. uvarum	4987	2759	2754	2376	3430	3968	4256

use to extract the locations of features, the protein sequences for genes, and the RNA sequences of transcripts.

`fagin` perfoms the following checking and cleaning steps:

- Assert that all columns have correct type (see **Table A.5**)
- Unify type synonyms (see **Table A.4**)
- Handle AUGUSTUS fields. The AUGUSTUS gene prediction program uses the tag 'Other' to represent the Parent relationship. If the `source` column of the GFF3 file (2nd column) is "AUGUSTUS", then the tag 'Other' will be converted to 'Parent' (with a warning that will be passed through `rmonad` to the user).

- Assert that each `Parent`, `ID`, and `Name` tag contains a unique value. In the GFF3 spec, a tag can be associated with a comma-delimited list of values. `fagin` currently does not support this and will raise an error if this case is found.
- Treat `Parent` tags with a value of '-' as missing.
- Handle unnamed fields. If no `ID` is given, but there is one untagged field, and if there are no other fields, then cast the untagged field as an `ID`. This is needed to accommodate the irregular output of AUGUSTUS.
- Assert parent child relations are correct.
  - If a feature links to a `Parent`, then the `Parent` must exist in the GFF
  - All `Parent IDs` must have either type `gene` or `mRNA`
  - All `CDS` and all `exon` must have a `Parent` (`gene` or `mRNA`)
  - All `mRNA` and `IDs` must be unique

Table A.4 GFF type (3rd column) equivalence groups. According to the GFF3 specification https://github.com/The-Sequence-Ontology/Specifications, names for the 3rd column of a GFF3 file should contain the names or IDs of elements from the Sequence Ontology [2]. Based on this, we coalesce members from the equivalence groups described in this table. The names listed in the **right** column (equivalent terms) are converted to the name in the **left** column. `SO:XXXXXXX` terms are Sequence Ontology IDs. We merge the `mRNA` and `transcript` groups; these groups are technically different, but they are often used interchangeably in practice. Also we have merged the `exon` and the more specific `coding_exon` terms.

term	equivalent term
gene	`SO:0000704`
mRNA	`messenger_RNA`, `messenger RNA`, `SO:0000234`, `transcript`, `SO:0000673`
CDS	`coding_sequence` , `coding sequence`, `SO:0000316`
exon	`SO:0000147`, `coding_exon`, `coding exon`, `SO:0000195`

Table A.5 A GFF file is a tab-delimited file with optional comments ('#' initialized). The table must have nine columns with the types listed under column **Base Type**. Entries in some columns are optional ('.' indicates optional). The phase column is required to be given for all GFF. The "attr" column contains a semicolon delimited list of `tag=value` pairs. According to the GFF3 specification, values may be comma delimited lists, but `fagin` does not currently handle these lists and raises an error if a comma appears in one of the tags required by `fagin` (`Parent`, `ID`, or `Name`).

Column name	Base type	Optional	Notes and Restrictions
seqid	string	No	all IDs present in genome
source	string	Yes	
type	string	No	
start	integer	No	
end	integer	No	$end \geq start$
score	numeric	Yes	
strand	$+\|-$	Yes	
phase	$0\|1\|2$	Yes	Required for CDS features
attr	tag-value list	No	

A.4 `fagin` data extraction from GFF and genome files

Get mRNAs. Given the GFF and genome, the extraction of the transcripts (mRNAs) is fairly straightforward. The sequences of all exons are extracted and concatenated. If the sense of the mRNA is negative, the result is then reverse transcribed.

Get proteins. Extracting the protein coding sequences from the genome given the GFF is slightly more involved. The GFF records all Coding Sequences (CDS) and associates each with a parent (an mRNA or gene feature). The CDS may be spread across many exons, and thus be a list of DNA intervals. These DNA intervals can be extracted from the genome and pasted together to form the full CDS. However, there is some nuance to this step. First, if the mRNA is negative sense, the CDS must be reverse transcribed. A more difficult case arises when the initial interval of the CDS does not begin in the correct reading frame. This can happen, for example, when part of the gene model is missing from the assembly. So the first interval in the CDS may begin on the 2nd or 3rd position on the codon. The 'phase' column of the GFF stores the number of nucleotides

that must be subtracted from the beginning of a CDS interval to read the first complete codon. `fagin` stores the phase data and will trim all models that start in a non-zero phase. Thus, partial protein models are allowed.

Once the CDSs, or partial CDSs, have been extracted, they may then they can be translated. `fagin` uses the `translation` function from the `Biostrings` package of the Bioconductor project. By setting `if.fuzzyi.codon="solve"`, it perform a "fuzzy" translation where codons with ambiguous nucleotides (e.g. N for unknown base or Y for pyrimidine) will either be translated as X (if more than one amino acid matches the pattern) or a specific amino acid (if only one amino acid matches). The resulting proteins are given the name of their parent and stored for future use.

Get ORFs in mRNAs and the genome. `fagin` identifies ORFs in the mRNAs and across the entire genome. ORF identification is limited to the mono-exonic case (i.e. splice sites are not searched for). This is often reasonable since young genes tend to have few or no introns (though there are notable exceptions to this trend [3]). `fagin` uses the Bioconductor ORFik package [4] to identify the longest, uninterrupted ORF for each stop codon in the genome (or mRNA). For the genome (but not the mRNA) it search both strands. The start and stop codons can be set by the user (the `fagin` default is `START=ATG` and `STOP=TAA,TGA,TAG`). The minimum ORF length can also be set by the user, with the default being 30 amino acids.

`fagin` currently use the standard gene table for all genes. This would cause problems in animal and fungi mitochondria and other cases where non-standard gene codes appear (plant organelle genomes use the standard gene code).

A.5 `fagin` homology inference statistics

`fagin` calculates sequence similarity for proteins and DNA through Smith-Waterman alignments of the query features of the focal genome (e.g., the gene sequence, spliced mRNA sequence, or translated coding sequence) against the features on the target

genome that overlap the search interval. These alignments provide a score, but no direct measure of statistical significance. To estimate the statistical significance of sequence matches, `fagin` infers p-values from an estimated false-positive distribution.

To simulate the false positive distribution, `fagin` starts with *reversed* query sequences. The sequence reversal erases the homology signal while preserving the sequence composition and site-dependencies (assuming site-dependencies are symmetric). For DNA, the order of nucleotides is reversed but not complemented. The idea of using reversed sequences as a control has been explored in the past [5, 6]. For proteins, the order of amino acids is reversed. Reversed proteins have been used as a control to test sequence masking algorithms [7]). There is no natural process that will reverse the order of codons in a coding sequence (since this would require many independent tri-nucleotide inversions), or to reverse the order of bases in a nucleotide sequence (inversions would also take the complement of the bases).

`fagin` next compares randomly selected query/target pairs of sequences. For protein searches, k query proteins are randomly sampled with replacement. For each sampled query protein, one target is randomly selected for each target that overlaps a search space on the target genome. For nucleotide searches, the k query genes are each searched against a random search interval. The alignments of the reversed query genes against the randomly chosen target sequences provides scores that approximate draws from the false positive distribution.

The raw Smith-Waterman scores need to be adjusted to account for search space size. The search space size is the product of the length of the focal sequence and the summed lengths of all target sequences. To account for search space size, `fagin` replaces the score with an adjusted score:

$$S_i' = S_i + b_0 + b_1 log(m_i \sum_{j=1}^{J} n_j) \tag{A.1}$$

where S_i' is the adjusted score, S_i is the original raw alignment score, m_i is the length of the focal amino acid sequence, n_j is the length jth of the target sequence, and J is the total number of target sequences that will be searched. b_0 and b_1 are the regression coefficients for the robust regression of the raw scores on the search $log(mn)$ where mn is the search space size. Robust linear regression was performed using the R function `L1fit` from the `L1Pack` package which implements the Barrodale-Roberts algorithm for L1 linear approximation [8].

Once adjusted scores are obtained against random search intervals, `fagin` fits the simulated highest scores for each query gene to a Gumbel distribution (a model of maximum values) using the R packages `fitdistrplus` [9] (using the maximum goodness-of-fit estimation with Cramer-von Mises distance), and calculates one-sided p-values for observed hits from this distribution. Finally, the p-value for each query is adjusted for the number of target sequences (across all species) that it is searched against (using the Holm method by default).

`fagin` uses the local Smith-Waterman algorithm, as implemented in the Bioconductor Biostrings package [10], to align each query gene to each of the similar sequences found in its target-genome search interval(s). For protein sequences, `fagin` uses the BLOSUM80 substitution matrix by default (the user may choose a different one). Nucleotide sequences are aligned with a local Smith-Waterman algorithm with a gap opening penalty of 10 and gap extension penalty of 4.

Bibliography

[1] Arendsee, Z. *et al.* (2019) synder: inferring genomic orthologs from synteny maps. *bioRxiv*

[2] Eilbeck, K. *et al.* (2005) The sequence ontology: A tool for the unification of genome annotations. *Genome biology* 6, R44

[3] Yang, Z. and Huang, J. (2011) De novo origin of new genes with introns in *Plasmodium vivax*. *FEBS Letters* 585, 641–644

[4] Labun, K. and Tjeldnes, H. *ORFik: open reading frames in genomics.* R package version 1.1.14

[5] Schwartz, S. *et al.* (2003) Human-mouse alignments with BLASTZ. *Genome research* 13, 103–107

[6] Grabherr, M.G. *et al.* (2010) Genome-wide synteny through highly sensitive sequence alignment: Satsuma. *Bioinformatics* 26, 1145–1151

[7] Frith, M.C. (2010) A new repeat-masking method enables specific detection of homologous sequences. *Nucleic acids research* 39, e23–e23

[8] Barrodale, I. and Roberts, F.D. (1973) An improved algorithm for discrete l_1 linear approximation. *SIAM Journal on Numerical Analysis* 10, 839–848

[9] Delignette-Muller, M.L. and Dutang, C. (2015) fitdistrplus: An R package for fitting distributions. *Journal of Statistical Software* 64, 1–34

[10] Gentleman, R.C. *et al.* (2004) Bioconductor: Open software development for computational biology and bioinformatics. *Genome biology* 5, R80

APPENDIX B. `metaoku` AND SELF-ANNOTATING NESTED DATA (SAND)

Zebulun Arendsee and Eve Syrkin Wurtele

Previously submitted to the Oxford DATABASE journal (and rejected). We have since abandoned the SAND specification since a similar idea is being developed by the frictionless data group (https://frictionlessdata.io/) and DataHub.

Abstract

Outside the gated communities of formalized databases, scientific data are often untidy and poorly documented. The lack of a consistent strategy for annotating and representing data reduces readability and complicates downstream analysis. To ease data access, we propose a standard directory organization that encapsulates data and documentation into modular units: Self-Annotating Nested Data structures (SAND). Creating these structures, or converting from existing structures, requires no expertise from the data generator beyond an understanding of their data. We showcase the utility of SAND with `metaoku`, a browser-based tool that enables the user to explore, share and retrieve data.

B.1 Introduction

Large, homogeneous data are well suited to monolithic databases. Databases like the NCBI Short Read Archive, for example, provide specialized tools for retrieving and analyzing data in standardized formats [1]. Great strides have been made to ensure

researchers submit their data to these databases. Managers of such big data build sophisticated standards out of necessity, this is not the case for the producers of small data.

Producers of small, non-specified data tend to arrange and document their tables using homespun, and often arbitrary, systems. The countless spreadsheets cluttering researcher's personal desktops, journal's supplementary material, and lab websites lack a common scheme. They are stored in files of diverse types (e.g. CSV, XLSX, MS-DOCX, or PDF) and formats vary even within a type. Locating and deciphering documentation is often difficult for a human and nearly impossible for a machine.

Tabular data, which we define as n-by-m grids of values where the first row contains column names, is perhaps the most universal, useful, and accessible of all data structures. It may be stored in many formats of varying human and machine readability; these may be classified roughly as structured, unstructured and semi-structured [2].

Unstructured data formats include human-readable formats such as Excel, PDF, Word, and HTML. Reliable extraction of the tables nested in these documents is possible but complicated [2]). These formats may contain freeform human-readable annotations, but distilling them computationally from the textual context is a difficult problem and an active area of research [3].

Structured data formats include machine-readable, formal databases. These allow powerful programmatic access to the technically savvy. Relational database managers, like SQL and its relatives, order groups of tables following a schema and allow complex search and merge operations. NoSQL database managers deviate from the table paradigm and express data as linked documents. This power comes at a cost, however, since there is an upfront cost to building such systems and a further cost to learning how to use them. Formal databases are overkill if 1) the data are neatly represented as tables, 2) the users need all the data (i.e. no need for queries), and 3) the tables are fairly independent (i.e. no need for table unions and similar SQL operations). A researcher

wishing to share a few tables of experimental results with a collaborator does not need database.

Semi-structured data formats include tables stored as raw text, with columns delimited by a special character (usually a comma or TAB). While this format is immediately readable by humans and machines, documentation cannot be added without compromising machine-readability. The annotations required to understand tabular data include the overall context of the table and the meaning of each column. Short column names are convenient in printed tables and scripts, however the short names may not provide sufficient documentation. Column names may be long, containing whole sentences, but this is problematic since: 1) they decrease human-readability by increasing the column width; 2) they decrease machine-readability since they are likely illegal or awkward variable names (for example, R replaces most non-alphanumeric characters with periods); and, 3) if the data are plotted, they may result in labels of impractical length. Documentation may be included in commented sections preceding the header, but this approach 1) is not supported by many tools, 2) is not expected by most users, 3) limits the expressiveness of the annotation (cannot include graphics), and 4) requires viewing and editing potentially large files. This lack of coherent data management hampers data sharing and secondary analyses.

Formatting and documenting a table is only the first stage of organization. Multiple separate tables, and the relationships between them, must also be stored. The most familiar, universally understood, and portable way to organize files is in a directory tree. Curators of data frequently share files of various types via FTP directories (e.g. the TAIR archive: ftp://ftp.arabidopsis.org/home/tair).

In this paper, we describe a strategy for coupling data and all necessary metadata in a simple file directory tree. Acceptance of such a common standard would create a niche for tools that automatically analyze, validate, describe and distribute nested datasets. It provides a logical and transparent means of structuring data for our own use and for

sharing with the broader community. The specification is very light, requiring only small changes in data management for large improvements in documentation and portability.

B.2 Description

B.2.1 SAND: Self-Annotating Nested Data

We propose storing data in conventional directories. Each table of data is in a dedicated folder with two associated annotations: a README containing general information about the data and a COLUMN tabular file with one row describing each column in the main table. This folder may be nested, along with an arbitrary number of sister folders, in a parent folder. The parent folder contains a README describing its children and may in turn be the child of a broader set. In this way, data are encapsulated as modular units containing everything needed to understand them, hence we term them Self-Annotating Nested Data (SAND) structures. For a simple example of this structure (**Figure B.1**).

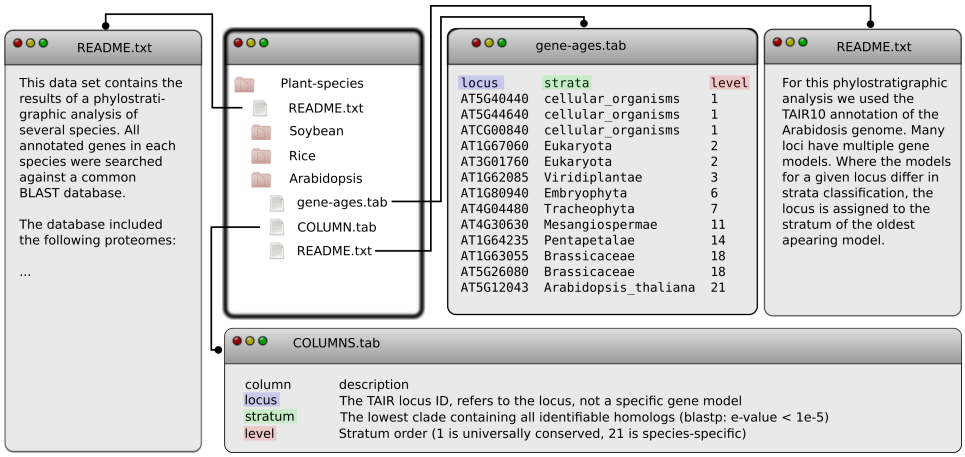


Figure B.1 SAND example. All files with the .tab extension are TAB delimited. For aesthetic purposes, the TABs have been padded with enough space to make even columns. Note that this is not legal in SAND. The "README" and "COLUMN" are reserved filenames in SAND format. Legal delimiters for the tabular files (both the data and the COLUMN file) are commas (.csv) and TABs (.tab, .tsv or .txt). Having the README be in a text format allows easy recursive keyword searching, but is not required. Since it is intended to be a human readable document, any format is legal.

The README should contain enough information for a human reader to understand where the described data came from (for external nodes) or what ties together the child datasets (for internal nodes). If the data table is the output of a program or pipeline, the README might describe the parameters, software versions, and inputs the reader would need to replicate the experiment. If the data table contains experimental results, the README should describe the methods. For an internal node holding a collection of datasets associated with a study, the README should contain a reference to the study.

We require a COLUMN table and not a ROW table because we assume the data are in "long" format, with columns as variables and the rows as observations. This format is usually more conducive to analysis [4].

A guiding principle in the development of SAND is the minimization of surprise. Every element has a place and is in its place. Documentation is in the README, exactly where it is expected to be. Columns names are in the COLUMN file. Cryptic clutter is absent entirely.

This approach is simple, universally portable, human and machine readable, and easily distributable (e.g. data trees can be shared as ZIP files). The README and COLUMN files in a data tree should be sufficient to allow the reader to fully understand, and if applicable, replicate, the data.

A limitation of the approach is that, at least as currently specified, it does not support non-tabular formats. Another limitation is that a tree structure allows a node to have only one parent. If it logically belongs in multiple places in the tree, it must exist in multiple copies. Another solution would be to allow soft links, but this is not portable and can lead to infinitely recursive file structures.

B.2.2 `metaoku`: An interface to SAND

`metaoku` is a browser-based tool designed to interface with SAND datasets and allow them to be shared and visually explored. It allows retrieval of SAND datasets as ZIP

files, viewing of the metadata, powerful searching and filtering of selected tables (via a wrapper for the JavaScript library DataTables (https://github.com/rstudio/DT)), automatic plotting of up to three variables, and a highly customizable plot builder (**Figure B.2**).

Automatic plotting requires a sophisticated type system. `metaoku` parses all columns into one of five types: numeric, categorical, textual, sequence (e.g. an amino acid sequence), or character. When the autoplotting function is called, `metaoku` looks up an appropriate function in a 6X6X6 array (the five datatypes plus a missing axis). Comparisons that are nonsensical, impractical, or not yet implemented will gracefully display an empty window.

`metaoku` is designed to be decentralized. Individuals, labs, and organizations can host their own instances. They may use an existing flavor, or they may design one tailored to their aesthetic desires and data-dependent requirements. The simplest usage of `metaoku` is as a stand-alone tool where users explore their own data (SAND formatted directories or single tabular files). Alternatively, it may be used as a graphical interface to an archive where SAND formatted data are packaged on the server and explored or retrieved by the client. The final possibility is a fully open setup where anyone with access to the server may read and write to the SAND project.

We chose to develop `metaoku` using Rstudio's Shiny framework (http://CRAN.R-project.org/package=shiny) This allows fast development and easy statistical and graphical operations. A downside is that Shiny apps can be slow, do not scale well with large data, and limits the number of concurrent users (at least in the free version). Given sufficient cause, we may retool using a faster, more scalable technology.

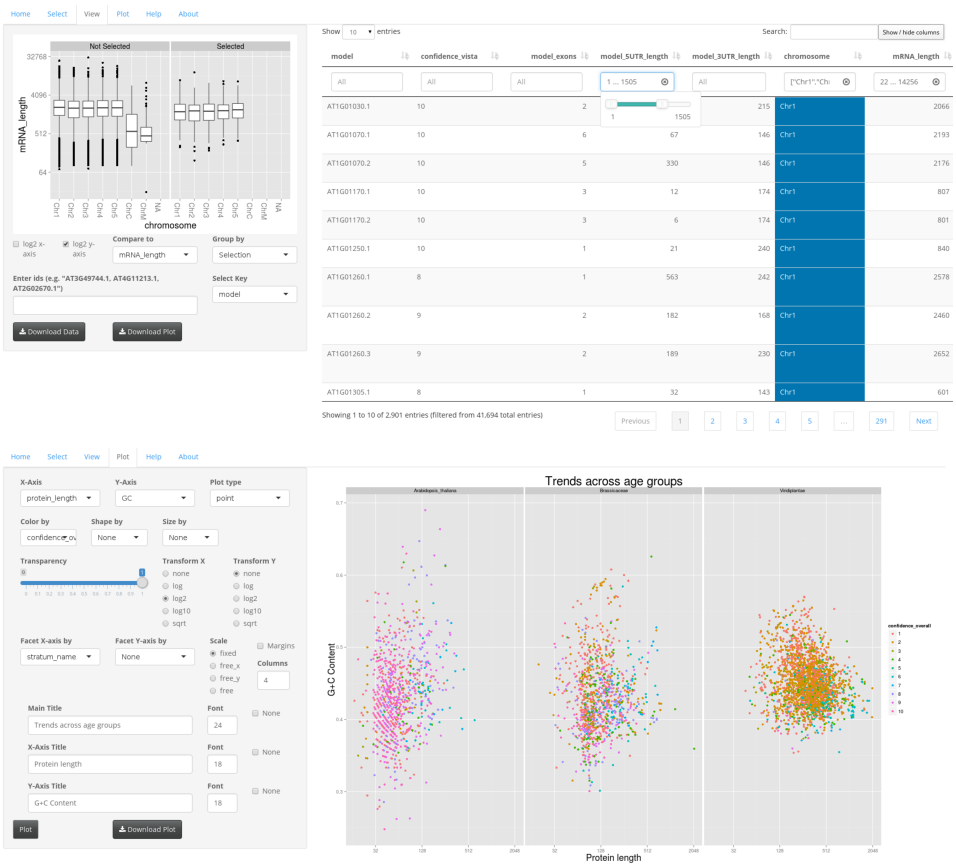


Figure 2

Figure B.2 Metaoku example. **(above)** The View tab is the main interface to an individual table of data. Columns selected in the spreadsheet window will be automatically plotted on the top left. Rows may be sorted by clicking the arrows beside the column names. Rows may be filtered by adding criteria to the boxes beneath or column names. Rows may additionally be filtered by pasting ids into the box on the middle left (a sample entry is automatically generated and dynamically updated when the key changes). The "Compare to" menu compares the selected column in the spreadsheet window to a second column. The "Group by" menu adds a third dimension. Metaoku parses all columns into one of five datatypes: numeric, categorical, textual, sequence (e.g. an amino acid sequence), or character. Metaoku looks up an appropriate function in a 6X6X6 array (the five datatypes plus a missing axis). Comparisons that are nonsensical, impractical, or not implemented gracefully display an empty window. The buttons on the bottom left allow downloading of the filtered data table or the current plot as a PDF vector image. The Plot tab **(below)** plots to be customized and tuned to the exact desires of the user. The options in the left sidebar are highly dynamic. The user first selects an X-axis, then they select Y from the subset of variables that are compatible with the datatype of X, finally they select a plot type. Once the plot type is selected, all the options for that plot type appear. The plot updates only when the user hits either the "Plot" or "Download Plot" button. The plot downloads as a PDF vector graphic.

B.2.3 Applications

The simplest application would be a personal or small group instance of `metaoku` that shares a working set of data. For example, `metaoku` could function as a lab notebook, where tables of experimental data are uploaded in SAND format.

The SAND structure can enhance reproducibility of computational research [5] by tightly coupling data and annotation in a standard fashion. Also scientific journals can increase the transparency of the research they publish by hosting `metaoku` instances to interactively share the data. Furthermore, SAND could be used as a pipeline output format, where the software version, dependency versions, and input parameters are all recorded in the README.

The conversion to SAND organization will create a niche for new data interfaces and tools. A deeply nested SAND structure containing hundreds of individual tables could be parsed into a single document by recursive assimilation of folder names and the content of README and COLUMN files.

Our first implementation of `metaoku` may serve as a template for an ecosystem of interfaces. For example, `metaoku` can be extended to statistical analysis, or optimized for a particular type of analysis (e.g. transcriptomics or metabolomics), or adapted to network analysis and geographical mapping.

B.2.4 A case study: genomic distribution of young genes

SAND and `metaoku` originated from our desire to better share and explore trends across a dataset of annotations for *Arabidopsis thaliana* genes [6]. We inferred the age of each gene by estimating the narrowest clade into which all homologs of the gene cluster (a technique known as phylostratigraphy [7]). We then merged many gene traits from many sources into a single table. Keeping track of the origins and contents of all these columns became a reoccurring problem. Many of the columns have complicated natures resistant to succinct expression in a header field. We also frequently revised this dataset,

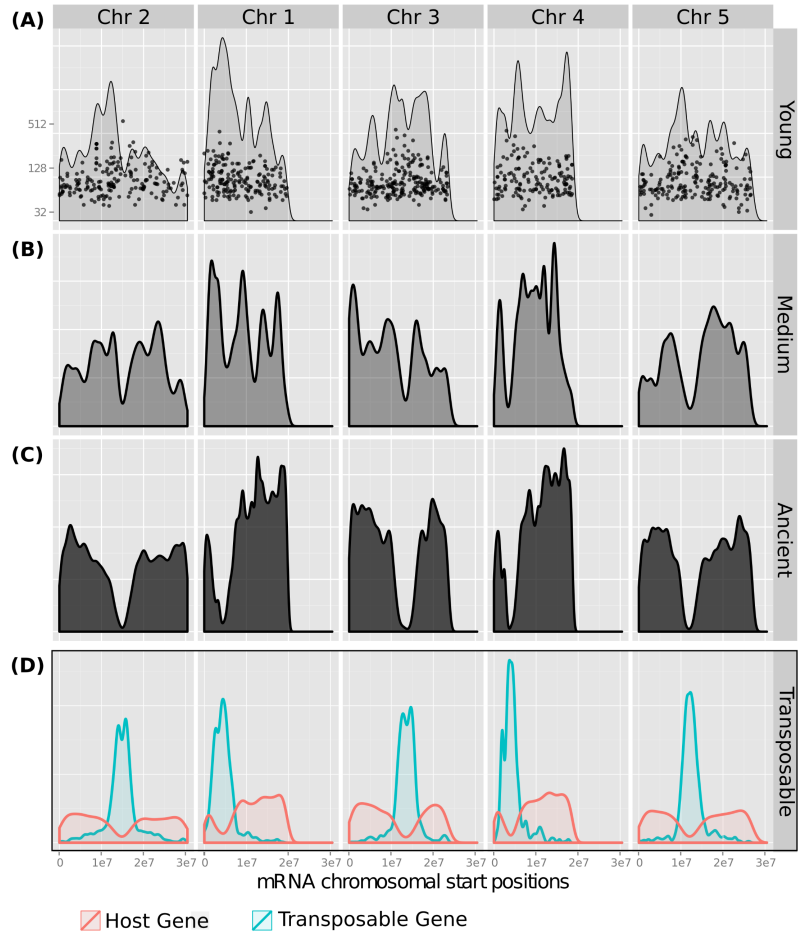


Figure 3

Figure B.3 Metaoku example. **(A)** Young, recently evolved genes appear to be fairly uniformly distributed across the chromosomes. The points correspond to individual genes with the length of the protein product on the y-axis. The older genes **(B-C)** exhibit deep valleys of low density that are centered on each chromosome's centromere. Interestingly, the transposable element density **(D)** is inversely correlated with host gene density. There may be a correlation between transposable density and young-gene density, although visually estimating the significance of this trend is not easy. This figure is based on the output of three Metaoku plots designed in the Plot tab. The PDFs were downloaded and then merged and tweaked in the vector editor Inkscape.

resulting in many copies of uncertain origin. With `metaoku` we can share a single, well-annotated, version of this data. We can explore the data and quickly build high-quality visuals.

Our primary research interest is the evolutionary origins of very young genes. Many of these are believed to have arisen from non-coding sequence [8]. We are interested in testing the hypothesis that novel genes arise at vastly accelerated rates in regions of either fast recombination or loose transcription. Such "gene nurseries" would manifest as clusters enriched in species-specific genes. Formally testing this hypothesis will require a careful statistical analysis, but we can gain quick intuition into the problem by visual inspection. We loaded the most recent version of our data from [6] into `metaoku`. Then using the `metaoku` frontend to the R package ggplot2 (http://CRAN.R-project.org/package=ggplot2)(see **Figure B.2**), we mapped all the genes specific to the *A. thaliana* species, the Brassicaceae family, and all cellular life to their positions on the *A. thaliana* chromosome (**Figure B.3**). **Figure B.3** clearly displays the paucity of genes in the vicinity of the centromeres. This is not surprising. The *A. thaliana* genes, however, follow a vastly different pattern, appearing to be fairly evenly distributed across the genome. Transposable elements follow a distribution exactly opposite of the ancient genes, with sharp peaks in the centromere regions. Interestingly, there may be a correlation between transposable density and young-gene density. We can not make any definitive conclusions about gene nurseries based on these graphics, but they provide a context and intuition from which we can plot our next steps.

B.3 Future directions

The base implementation of SAND can be extended in several ways. The human-readable packet description could be supplemented with a machine-readable counterpart, perhaps in XML or JSON format. These machine readable descriptions could hold a schema for the table that relates it to other tables in the dataset, that is, they could hold

the map required to convert the data packet into a relational database. A data packet could also contain folders of code to analyze the data and output from such analyses. Another possibility is the extension of SAND to non-tabular data. For example XML data, where the FIELDS file would be replaced with a DOM.

B.4 Conclusion

SAND is a formal methodology for structuring data using only tools familiar to every researcher. Converting from idiosyncratic organizations of data and their documentation to SAND format requires merely a rearrangement of existing material. Yet this change is sufficient to permit both humans and machines to traverse the structure, processing data and its documentation, with minimal hassle.

`metaoku` allows researchers to survey datasets with ease, flipping through visual summaries of columns and their inter-relationships with a few clicks of the mouse. Whole SAND structures can be retrieved, or the data can be filtered and downloaded in part. By allowing users to upload data and annotations, conforming to SAND specifications becomes almost passive, since `metaoku` handles the SAND structure invisibly. Data producers may focus on describing their data without being bothered by the details of database management. Data consumers may retrieve full datasets, quickly read through their documentation, and proceed with analysis, wasting little time hunting for documentation or prying tables from messy Excel sheets. Data curators may glean semantic relations from the documentation and assemble the tables into relational databases. The benefit of compatibility with SAND-based tools, like `metaoku`, will encourage data producers to structure and annotate their data in a way that will benefit all within the community.

SAND, coupled with tools such as `metaoku`, will provide a way to more transparently and effectively share, explore and integrate the vast space of tabular datasets with non-specified schemata.

B.5 Code Availability

The `sandr` and `metaoku` source code is available on github at https://github.com/arendsee/sandr and https://github.com/arendsee/metaoku, respectively. The github page for `metaoku` displays the basic documentation contains links to interactive usage cases.

Author Contributions

ZA and ESW conceived `metaoku`. ZA implemented it and designed the SAND specification. ESW provided extensive feedback and direction for the functionallity of `metaoku`.

Bibliography

[1] Kodama, Y. *et al.* (2011) The sequence read archive: explosive growth of sequencing data. *Nucleic acids research* 40, D54–D56

[2] Khusro, S. *et al.* (2015) On methods and tools of table detection, extraction and annotation in PDF documents. *Journal of information science* 41, 41–57

[3] Adelfio, M.D. and Samet, H. (2013) Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment* 6, 421–432

[4] Wickham, H. *et al.* (2014) Tidy data. *Journal of Statistical Software* 59, 1–23

[5] Sandve, G.K. *et al.* (2013) Ten simple rules for reproducible computational research. *PLoS computational biology* 9, e1003285

[6] Arendsee, Z.W. *et al.* (2014) Coming of age: orphan genes in plants. *Trends in plant science* 19, 698–708

[7] Domazet-Lošo, T. *et al.* (2007) A phylostratigraphy approach to uncover the genomic history of major adaptations in metazoan lineages. *Trends in Genetics* 23, 533–539

[8] Carvunis, A.R. *et al.* (2012) Proto-genes and de novo gene birth. *Nature* 487, 370–374

APPENDIX C. THE ICEHGR FAMILY

Zebulun Arendsee

This chapter was adapted from a class project in BCB568 (under Robert Jernigan and Guang Song); the results are preliminary. The ICEHGR family is one of several large families of highly repetitive, species-specific genes I found in large study of proteomes across Eukaryotic kingdoms. I found these families by accident while trying to determine whether protein sequences where "reversible". To address this question, I looked at amino acid triplets and their inversions, supposing that they should be of equal frequency if the protein sequences are reversible. To my surprise, I found that a few triplets appeared orders of magnitude more frequently than their inversions. Further, for several cases, the discrepancy appeared only in one species. Looking into this, I found, among others, the CEH triplet which led me to the ICEHGR family in *Micromonas pusilla*.

C.1 Background

The analysis of the first several full genomes revealed the existence of long open reading frames (ORFs) that are unique to narrow clades. These young genes tend to have short lifespans [1], but those that manage to become fixed, may be preserved through speciation, becoming genes unique to large clades. Genes can be stratified by age by identifying the most evolutionarily distant species with a homolog and tracing back to the common ancestor. In this way, it is possible to see proteins at various ages and map the path proteins follow as they develop from random orphans to highly optimized, mature genes.

Perhaps the most obvious trend across time is in protein length. Orphans tend to be very short, for example, the median orphan in Arabidopsis is only about 57 residues in length, compared to the overall median length of 350 [2]. Of course there are several ways in which a protein can gain new material (addition of new exons, mutations in STOP codons, insertions), but here I will focus on one possibility: expansion of tandem repeats. There are several examples of repetition of a short sequence leading to powerful functions. One is the origin of an ice fish antifreeze protein by the tandem replication of a short microsatellite [3].

I have identified a gene family, which I call the ICEHGR family, that is unique to one of the two sequenced [4] subspecies of *Micromonas pusilla*. Members of this gene family are dynamic in two ways, 1) the genes themselves are replicating very rapidly (there are at least 80 members), and 2) they share an 18-residue repeat which appears to be actively expanding (with up to 20 repeats on each gene). This family may be a "protein nursery", exploring new functions by varying a common motif.

In this project, I will explore the primary sequence (the conservation of the repeat), the secondary structure, and finally the tertiary structure of this family.

C.2 Results

C.2.1 Classifying genes as ICEHGR

I first noticed the ICEHGR family when I found the amino acid triplet CEH was much more common than its reverse, HEC, in *Micromonas pussila*. Upon further inspection, I found the CEH words were heavily concetrated in less than 100 proteins. Furthermore, the CEH word was only a part of a larger, 18-residue motif, which I named ICEHGR after the most conserved motif.

To identify ICEHGR proteins I first performed a specialized blast of the ICEHGR motif against the *Micromonas pusilla* proteome (`blastp -word_size 2 -threshold 8 -evalue 99999 -comp_based_stats F`). This search identified any pattern roughly

matching the string "ICEHGR". Then I filtered out all proteins with fewer than 6 hits. Finally I removed any proteins longer than 1000 aa. This last step I performed solely to remove one enormous protein which I know, from inspection, does not contain a true ICEHGR repeat.

C.2.2 Gene family description

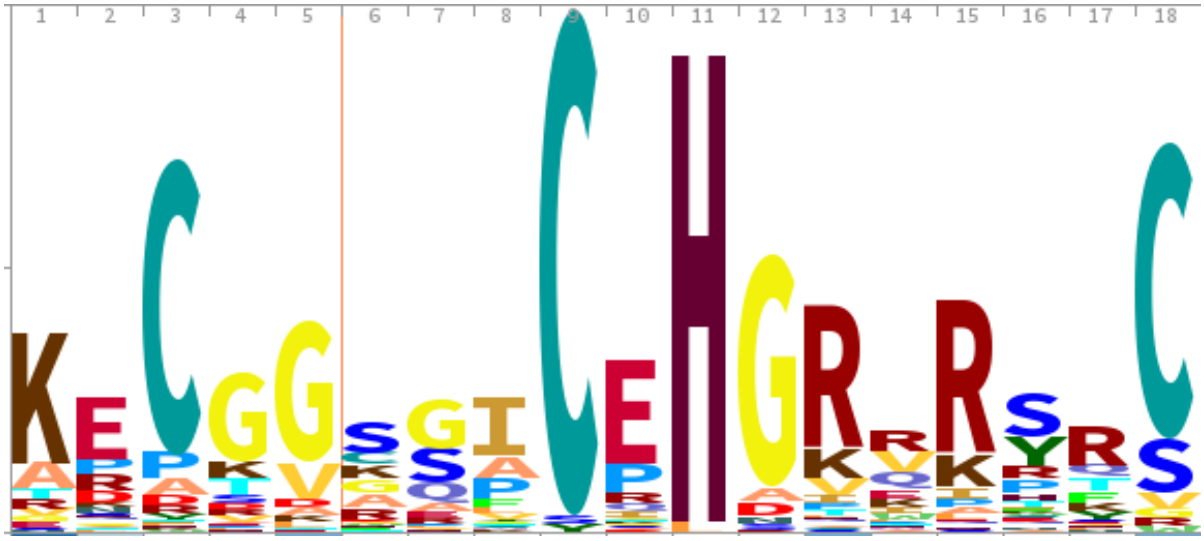


Figure C.1 Conservation levels for the 18 residues in the ICEHGR motif. This logo was built from the alignment of 880 ICEHGR motifs from the 80 ICEHGR proteins. I extracted the coordinates for the hits to the "ICEHGR" string from the BLAST results and then extended the region 7 residues upstream and 5 downstream and extracted the corresponding sequence from the protein FASTA file (with bedtools [5]). I then aligned these strings with MUSCLE [6], assessed their conservation with HMMER [7], and created the image with Skyalign (http://skylign.org/).

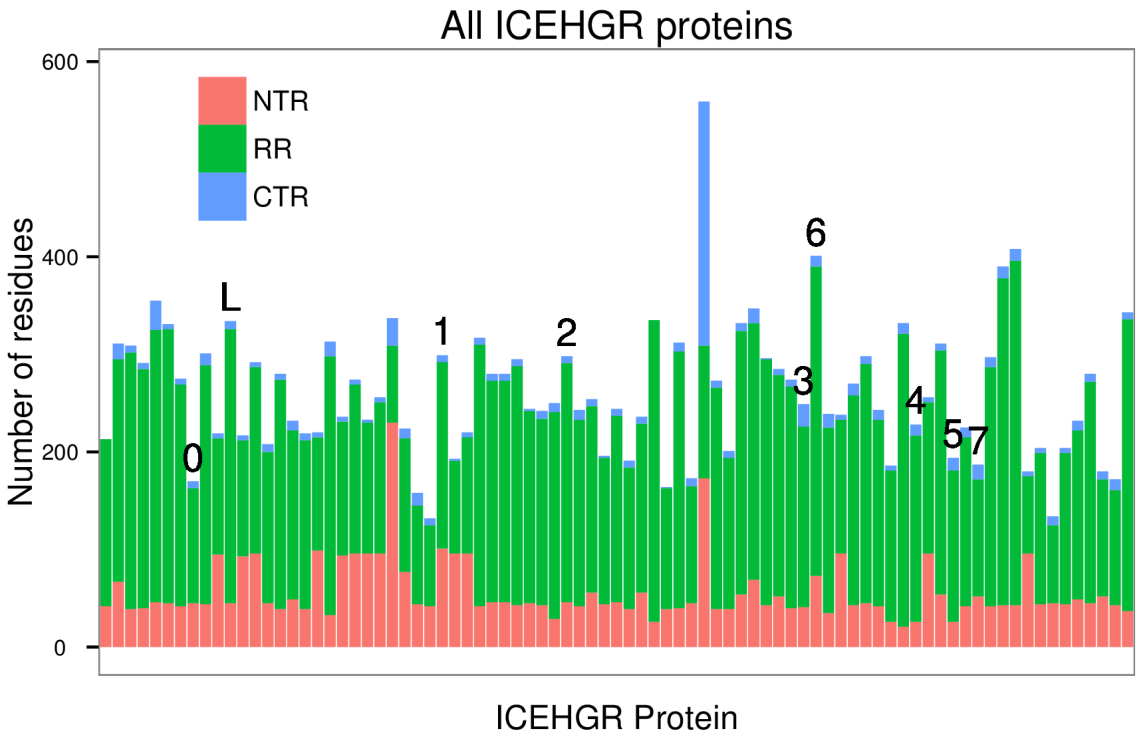


Figure C.2 Relative lengths of the n-terminal regions (NTR), repeat regions (RR) and C-terminal regions (CTR) for all 80 ICEHGR proteins. The numbers identify the proteins for which I have predicted structures.

C.2.3 Secondary structures

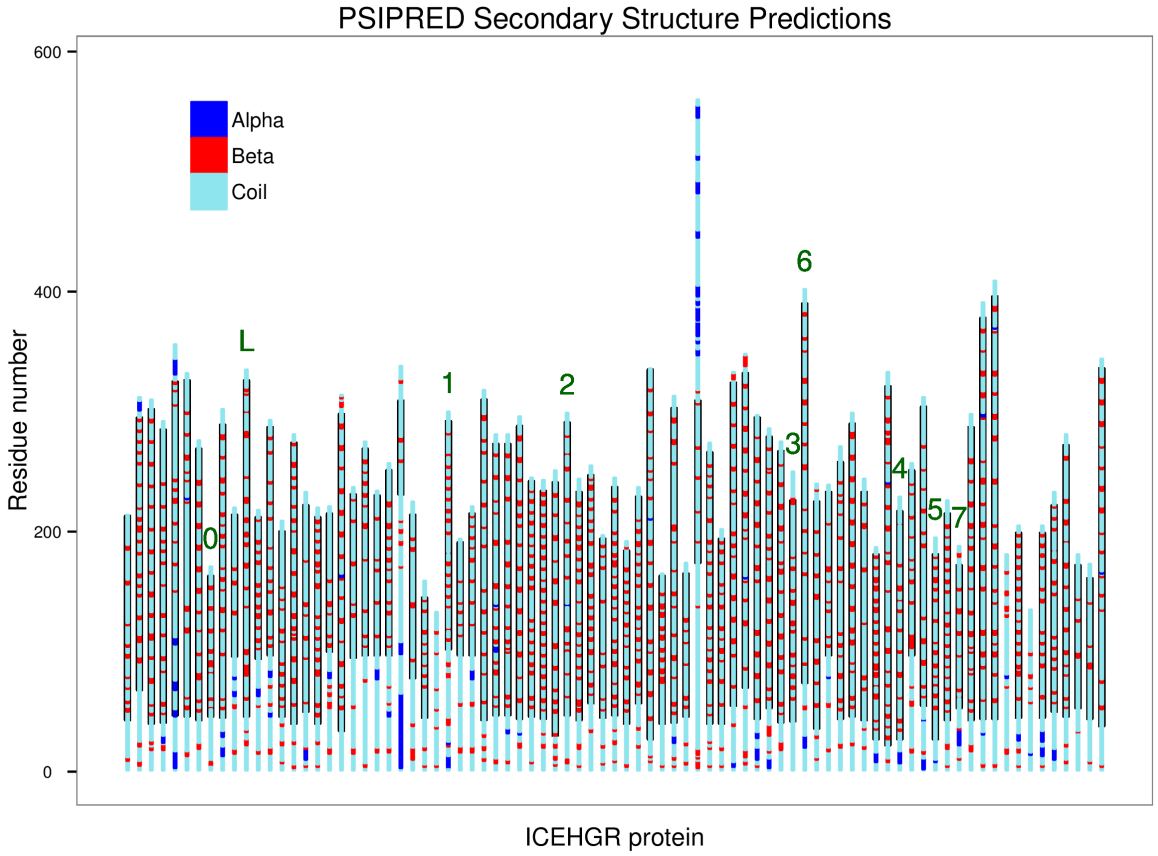


Figure C.3 Predicted secondary structure for all ICEHGR proteins. The regions flanked by black lines are the repeat regions. The red and blue bands correspond to β-sheets and α-helices, respectively. The light blue regions are coil (which includes turns). Structure was predicted with PSIPRED (using a custom BLAST database consisting of the full proteomes from plant, metazoan, and fungal species in the JGI genome databases).

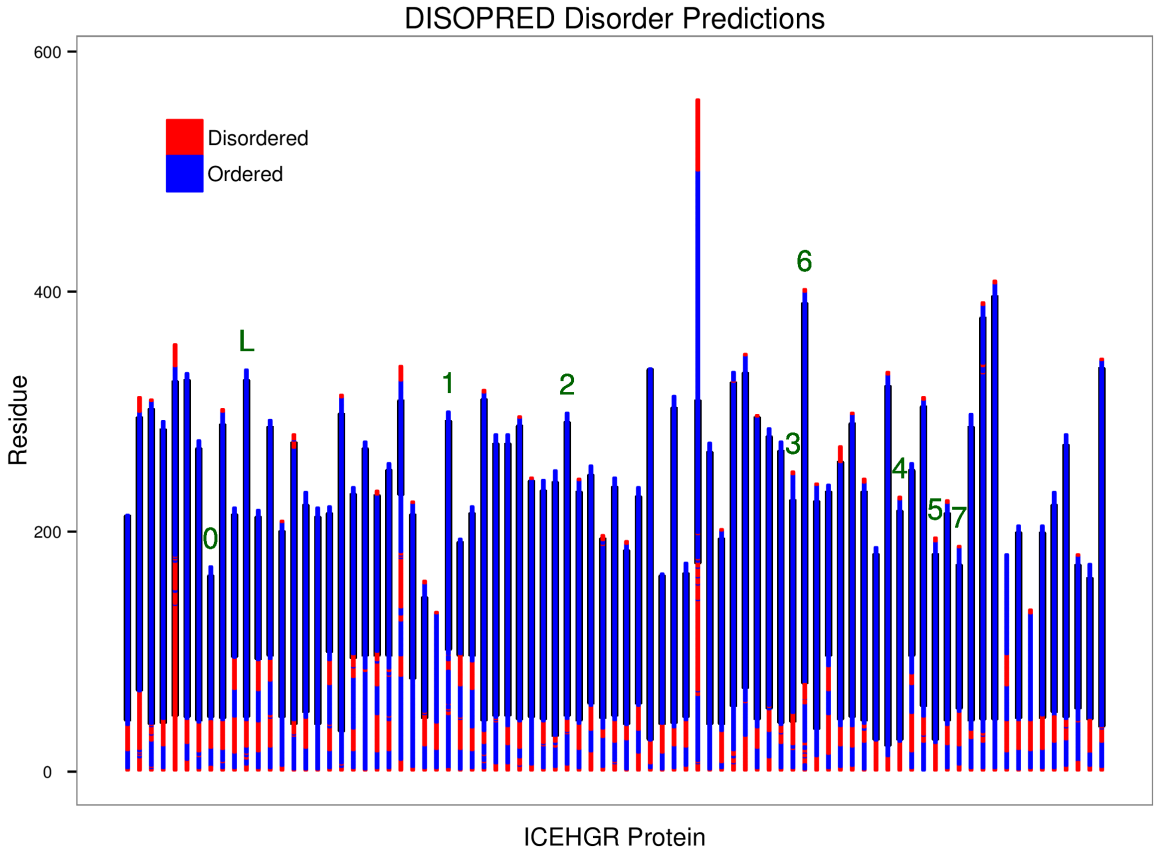


Figure C.4 Predicted regions of intrinsic disorder. Black flanked regions are the repeat regions. Red and blue are disordered and ordered respectively.

The repetitive regions of the ICEHGR genes are almost completely comprised of β-strands and coil. The periodicity of the β-strands varies greatly between proteins.

The NTRs are quite variable. They cluster into 3 large groups (`blastclust -i nterm.faa -L .4 -S 60`) with 28, 18, and 13 members. Each of the large clusters has around half of its members on one chromosome (chromosomes 9, 7, and 15, respectively). What they all have in common, however, is a stretch of disorder prior to the initiation of the repetitive region.

C.2.4 Tertiary Structures

I randomly selected 9 ICEHGR proteins for tertiary structural analysis. For eight of these I modeled the c-terminal region and the last four ICEHGR repeats and predicted 250 structures each using the Rosetta *ab initio* relax program (`-abinitio:relax -relax::fast -detect_disulfide_before_relax true -rebuild_disulf true`). For the ninth selected protein, I predicted 800 structures of the innermost 5 ICEHGR repeats.

My first approach to finding the best model, was to simply take the one with the lowest score.

Table C.1 ProQ Scores for lowest scoring decoys for each protein. Metrics are taken directly from the ProQ server [8] (http://www.sbc.su.se/ bjornw/ProQ). "L" is the long-running model (2095 decoys), 0-7 are the short-running models

model	n-decoys	LGScore	MaxSub	Comparisons
L	-0.379	800	-0.036	LGscore>1.5 fairly good model
0	-0.510	251	-0.021	LGscore>2.5 very good model
1	-0.991	251	-0.118	LGscore>4 extremly good model
2	-0.678	251	-0.053	
3	-0.742	251	0.005	MaxSub>0.1 fairly good model
4	-0.534	251	0.003	MaxSub>0.5 very good model
5	-0.432	251	-0.054	MaxSub>0.8 extremly good model
6	-0.389	251	0.001	
7	0.194	251	-0.006	

None of the models in Table C.1 seem acceptable based on ProQ. However the lowest scoring models are not always the optimal. A better approach is to group the models into clusters and then find the models at the "centers" of the largest few clusters. I performed this task using MaxCluster.

As seen in Table C.2, the scores are still not acceptable, but the centroids of the clusters do score better overall then those in Table 1. In spite of their bad scores, the structures for the top clusters are reasonably consistent (Figures C.5 and C.6).

Table C.2 Structures were clustered with MaxCluster using the Nearest Neighbor clustering algorithm. The centroid of each cluster is the single decoy with the greatest similarity to the most structures within the cluster. The size is the number of decoys in the cluster (capped at 50)

model	cluster	size	spread	Centroid LGscore	Centroid MaxSub
L	1	50	0.374	-0.653	-0.028
L	2	50	0.388	-0.387	-0.003
L	3	50	0.419	-0.438	-0.021
L	4	50	0.432	-0.216	-0.030
L	5	37	0.447	-0.209	0.023
0	1	50	0.379	-0.330	-0.063
0	2	50	0.400	-0.232	-0.012
0	3	19	0.432	-0.385	-0.038
1	1	50	0.439	-0.481	-0.080
1	2	23	0.436	-0.799	-0.099
1	3	10	0.410	-0.747	-0.121
2	1	50	0.413	-0.674	-0.098
2	2	29	0.430	-0.553	-0.061
2	3	17	0.432	-0.658	-0.112
3	1	50	0.342	-0.458	-0.049
3	2	50	0.385	-0.486	-0.067
3	3	49	0.426	-0.454	-0.060
4	1	50	0.370	-0.687	-0.091
4	2	50	0.375	-0.588	-0.058
4	3	46	0.430	-0.665	-0.073
5	1	50	0.508	-0.603	-0.054
5	2	36	0.550	-0.446	-0.093
5	3	20	0.515	-0.399	-0.014
6	1	50	0.412	-0.522	-0.083
6	2	14	0.406	-0.584	-0.072
6	3	12	0.433	-0.188	0.012
7	1	50	0.584	0.264	-0.021
7	2	32	0.578	0.368	-0.080
7	3	18	0.598	0.184	-0.060

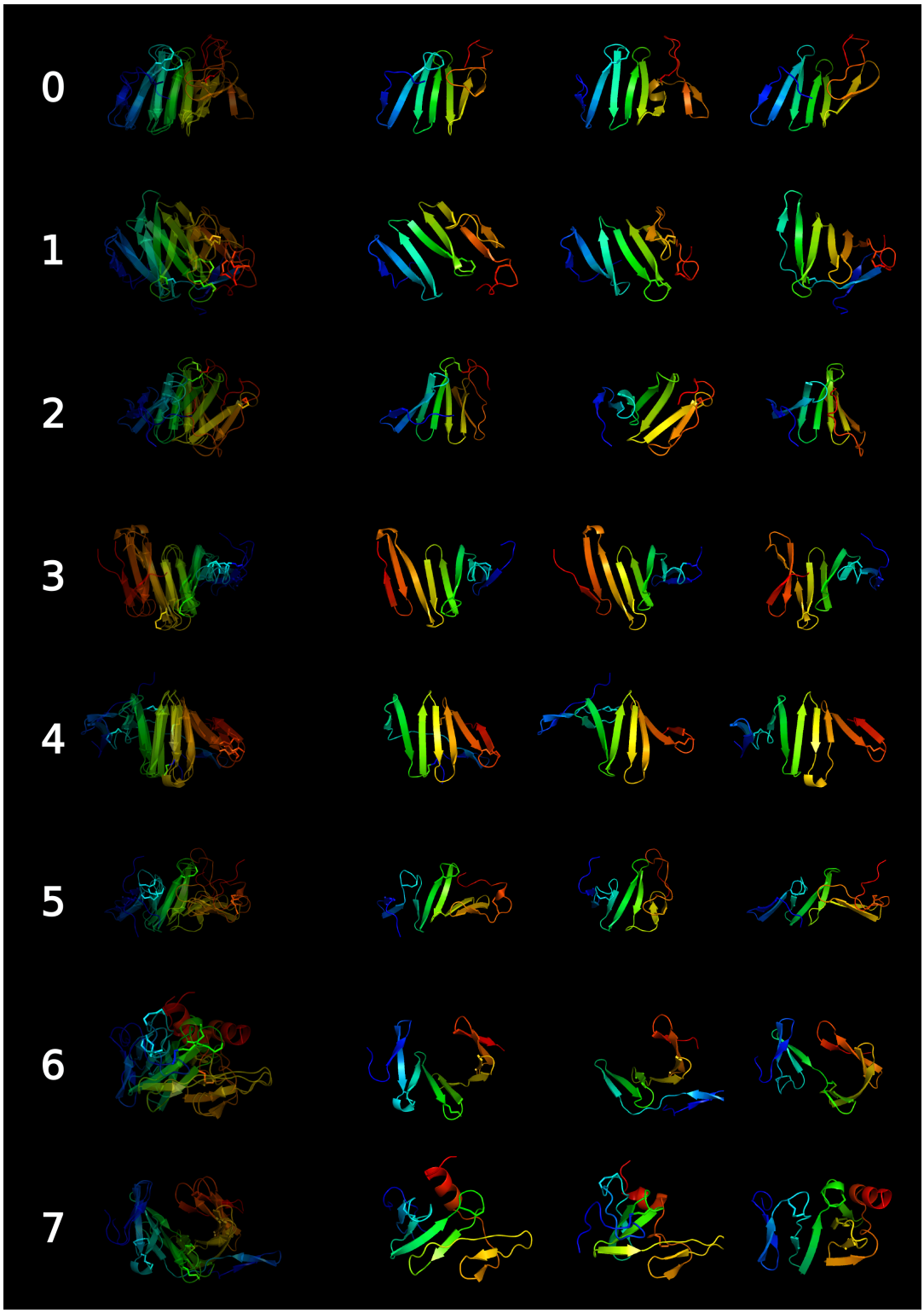


Figure C.5 8 short-run peptides, alignment on left, top three centroids from left to right

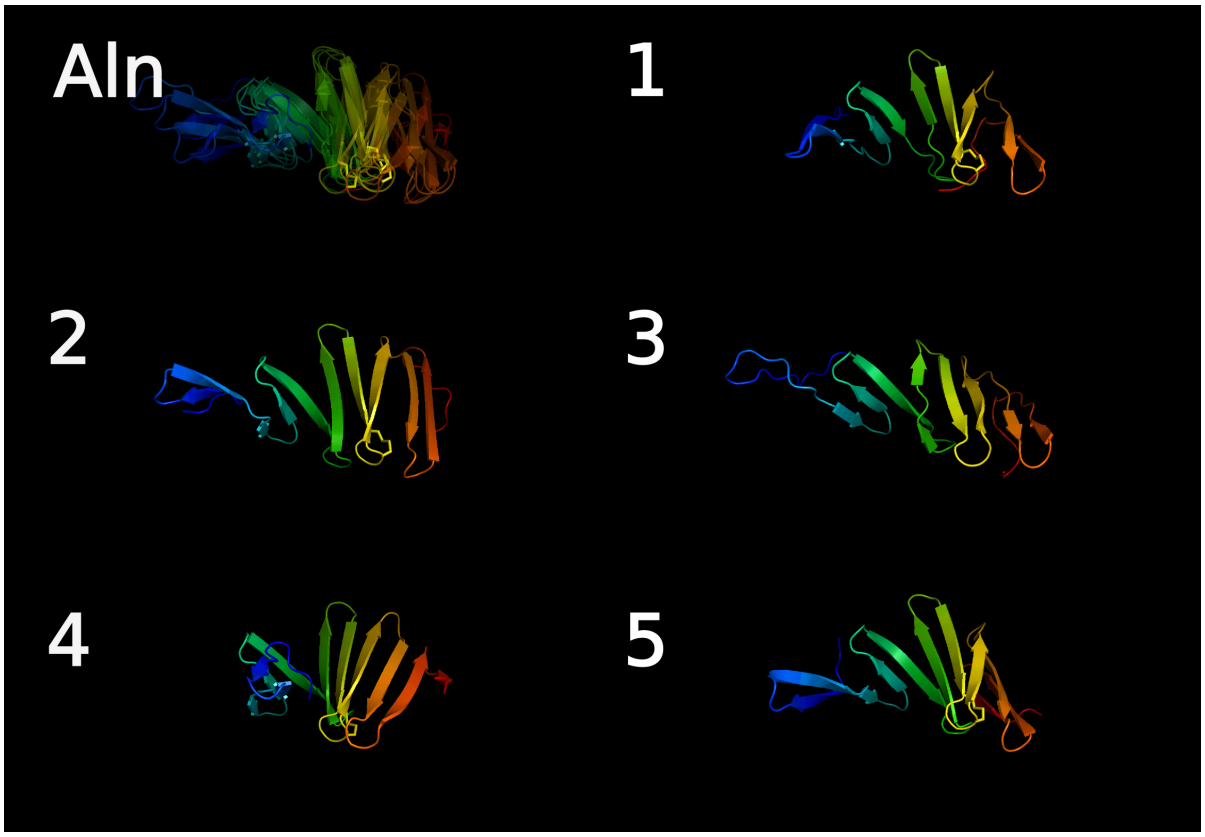


Figure C.6 The long running model, The top 5 centroids and their alignment

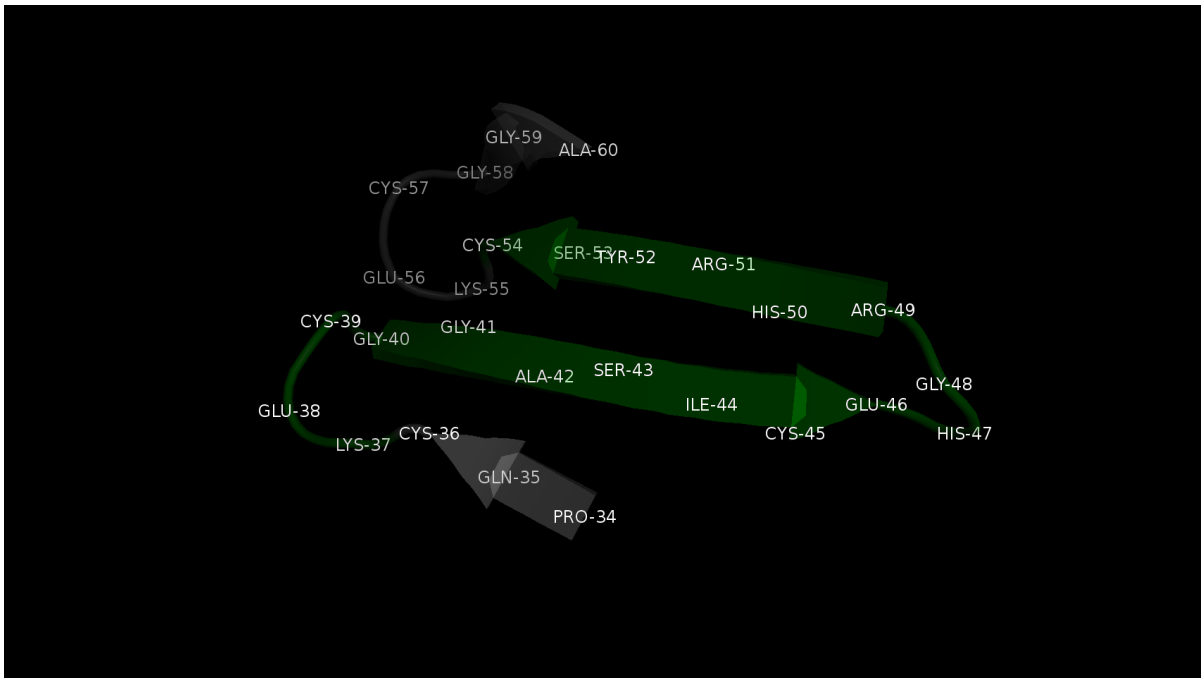


Figure C.7 An example of a single ICEHGR motif (highlighted in green). The HGR and KEC regions form the turns for most of the ICEHGR loops.

C.3 Discussion

I have uncovered the rough structure of the ICEHGR family. They consist of a short, ordered n-terminal region linked to a long series of ICEHGR repeats by a disordered region (Figures C.2 and C.4). The ICEHGR repeat regions are predicted to be ordered and consist mostly of β-sheets and turns (Figure C.3). Based on this, I suspect the ICEHGR repeat regions form an independently folding domain.

An ICEHGR repeat will, in an ideal case, form a single β-hairpin (Figure C.7) with a three residue turn. Many of these in tandem may form long β-sheets.

The role of the highly conserved cysteines (Figure C.1) remain elusive. They may act to stabilize the β-sheets by bridging across the turn, as is seen in many of the models. Or they may bond with more distant cysteines, forming interactions to large to be seen by my 4-5 repeat models.

So far I have only been able to model a few repeats. This is useful in elucidating the local fabric of the domain, but does not reveal the grander shapes an entire 10-20 repeat region might assume.

My modeling only hints at the amount of variation present within the ICEHGR population. In the future, I could build more rigorous models of the ICEHGR proteins. Perhaps I could inform the models with evolutionary information from motif conservation. Also, an extremely coarse grained approach, such as loop-modeling, might be effective in solving full ICEHGR domains.

Also I have barely touched on the identities of the N-terminal and C-terminal regions. Examining their structures and possible functions may hint at the functions entire ICEHGR genes.

Bibliography

[1] Palmieri, N. *et al.* (2014) The life cycle of Drosophila orphan genes. *ArXiv e-prints*

[2] Arendsee, Z.W. *et al.* (2014) Coming of age: orphan genes in plants. *Trends in plant science* 19, 698–708

[3] Chen, L. *et al.* (1997) Evolution of antifreeze glycoprotein gene from a trypsinogen gene in Antarctic notothenioid fish. *Proceedings of the National Academy of Sciences* 94, 3811–3816

[4] Worden, A.Z. *et al.* (2009) Green evolution and dynamic adaptations revealed by genomes of the marine picoeu- karyotes Micromonas. *Science* 324, 268–272

[5] Quinlan, A.R. and Hall, I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* 26, 841–842

[6] Edgar, R.C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 32, 1792–1797

[7] Eddy, S.R. (2011) Accelerated profile HMM searches. *PLoS Computational Biology* 7, e1002195

[8] Wallner, B. and Elofsson, A. (2003) Can correct protein models be identified? *Protein Science* 12, 1073–1086